НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Теплоенергетичний факультет
Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено:
Завідувач кафедри
_____ О. В. Коваль
(підпис) (ініціали, прізвище)
«___»_____2021 р.

ДИПЛОМНА РОБОТА

на здобуття ступеня бакалавра

спеціальності 122 «Комп'ютерні науки»

освітня програма «Комп'ютерний моніторинг та геометричне моделювання процесів і систем»

на тему: «Створення параметричних металоконструкцій в CAD системі PythonParts»

Виконав: студент 4-го курсу, групи ТМ-72
Коваль Ярослав Валерійович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник: доцент, к.т.н. Демчишин Анітолій Анатолійович _____
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Рецензент: _____ _____
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент: _____
(підпис)

Київ – 2021 року

Національний технічний університет України

“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет: теплоенергетичний

Кафедра: автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти: перший рівень

Спеціальність: 122 «Комп’ютерні науки»

Освітня програма: «Комп’ютерний моніторинг та геометричне моделювання процесів і систем»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О. В. Коваль
(підпис) (ініціали прізвище)
“___”_____2021р.

ЗАВДАННЯ

на дипломну роботу студенту

Ковалю Ярославу Валерійовичу

1.Тема роботи: «Створення параметричних металоконструкцій в CAD системі PythonParts»

керівник роботи: доцент, к.т.н. Демчишин Анатолій Анатолійович

(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”24” травня 2021р. № 1267-с

2.Строк подання студентом роботи: ”12” травня 2021р.

3.Вихідні дані до роботи: зформована модель армування стіни, палетка з заданими параметрами, відображувана у Allplan середовищі модель.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) проаналізувати проблему моделювання елементів у застосунку Allplan за допомогою PythonPart підсистеми, спроектувати архітектуру модуля, розробити програмний продукт, написати інструкцію використання для користувача.

5. Перелік ілюстративного матеріалу:1. Мета роботи. 2. Аналіз існуючих альтернативних BIM рішень. 3. Засоби розробки модуля. 4. Опис алгоритмів роботи модуля. 5. Інструкція користувача

Дата видачі завдання ”15” лютого 2021 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Затвердження теми роботи	15.02.2021	
2.	Вивчення та аналіз задачі	12.04-17.04.2021	
3.	Розробка архітектури та загальної структури системи	18.04-23.04.2021	
4.	Розробка структур окремих підсистем	19.04-5.05.2021	
5.	Програмна реалізація системи	23.04-16.04.2021	
6.	Оформлення пояснювальної записки	16.05-8.06.2021	
7.	Захист програмного продукту	12.05.2021	
8.	Передзахист	24.05.2021	
9.	Захист	18.06.2021	

Студент: _____ Коваль Я. В.
(підпис) (прізвище та ініціали)

Керівник роботи: _____ Демчишин А.А.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Дана робота має на меті створення модуля у системі PythonPart, яка є частиною BIM застосунка Allplan. Суть модуля полягає у проектуванні армування для будь-яких стін з різними формами та вирізами. Модуль створює периметральне армування, армування по площі та армування вирізів. Можуть використовуватися два шари стіни – передній та задній, які задаються з палетки. Шари як стін, так і вирізів, знаходяться за допомогою алгоритму пошуку точок, який основується на лінійній алгебрі. Також PythonPart палетка дозволяє користувачу гнучко налаштовувати потрібні параметри армування, такі як тип армування, діаметри стержнів, відступи від краю опалубки та ін. У роботі представлені прямі конкуренти системи Allplan. Також освітлені основні переваги застосунка Allplan, зокрема його підсистеми PythonParts. Розібрані необхідні інструменти та мотиви обрання цих інструментів для реалізації PythonPart модуля.

ABSTRACT

This work has the purpose of creating a module in the PythonPart system, which is part of Allplan’s BIM application. The essence of the module is the design of reinforcement for any wall with different shapes and cuts. Module creates perimeter reinforcement, area reinforcement and cuts reinforcement. Two layers of wall can be used, front and back, which are set from the palette. The layers of both the walls and the cuts are located by a point-finding algorithm that is based on linear algebra. PythonPart palette also allows the user to flexibly customize the necessary reinforcement parameters such as reinforcement type, bar diameters, recesses from the side of the form, etc. The work includes direct competitors of Allplan system. Also highlighted are the main benefits of Allplan’s application, in particular its PythonParts subsystem. The necessary tools and reasons for selecting these tools to implement PythonPart module were discussed.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

.pyd, .pyc – файли, з розширеннями відповідно .pyd або .pyc

UI – user interface, інтерфейс користувача

DLL – dynamic linked library, динамічно підключаєма бібліотека, використовувана у середовищі Windows

XML - Extensible Markup Language, формат текстових файлів для розмітки будь-чого.

JSON – Javascript Object Notation, формат текстових файлів для розмітки об’єктів, зокрема для мови програмування JavaScript.

IDE – integrated development environment або інтегроване середовище розробки, система, що дозволяє писати код, будувати проект, здійснювати дебаг та ін.

ВСТУП

Однією з найпоширеніших проблем на сьогодення є автоматизація будь-яких процесів за допомогою програмного забезпечення. Для будь-якого процесу автоматизації потрібен час, особливо для поглиблених відгалужень деяких галузей, наприклад, таких, як архітектура. За період розвитку інформаційних технологій людство винайшло чимало рішень, котрі виявилися незамінними у певних ситуація. Так і у архітектурі було створено масу корисних пристосувань.

Останнім часом активно створюються різні застосунки – як веб, так і десктопні, які дозволяють проектувати будь-які споруди, найрізноманітніших форм та призначень. Вони вони мають назву CAD (computer aided design, системи автоматизованого проектування) або BIM (building information modeling, інформаційне моделювання будівель) система та створюються виключно для нішових цілей. Хоча існують застосунки досить різних рівней складності, та зазвичай у комерційній діяльності компанії намагаються зайняти якогома більшу аудиторію, створюючи функціональність, яка покриває потреби більшості користувачів. Іноді виникають ситуації, коли користувачам потрібна не зайва функціональність, а та, яка б допомогла швидше вирішити конкретно їхні проблеми. Тому, у архітекрутних BIM застосунках різними шляхами створюються автоматизовані підсистеми, які призначені для пришвидшення виконання невеликих завдань.

Десктопна система Allplan не являється виключенням архітектурних BIM систем. Виконуючи свою роль, як програма проектування будівель будь-яких форм, мостів, цивільних споруд, вона надає доступ до широкого інструментарію архітектора, що дозволяє вирішити більшість архітектурних проблем. Хоча цей додаток має досить широкий спектр можливостей, але завжди можна знайти речі, які доводиться робити вручну. Тому з'явилася потребу у автоматизації деяких процесів, одним з яких стало армування стін.

Армування – це заготовка деяких форм з арматури для створення монолітної залізобетонної металоконструкції. Отже автоматизація армування полягає в

розрахунку потрібних форм, використовуючі дані, які часто використовуються користувачем.

Також в систему Allplan є інтегрована система PythonParts, яка дозволяє за допомогою Python модулів створити будь-який елемент, який пов'язаний з конструюванням. Тож для рішення поставленого завдання потрібно створити інтерактор на основі PythonParts системи, який, на основі заздалегідь заданих змінних, буде обчислювати розташування та параметри арматури на стіні. Такий підхід спростить розширення самого модуля, так як він буде створений на верхньорівневій мові програмування, та інтеракцію користувача з застосунком, тобто зменшить вхідний поріг для початкових користувачів, які ще не мають досвіду у армування стін.

Метою роботи є розробка алгоритмічної та теоретичної бази для армування стін з вирізами (двері, вікна, бокові вирізи) в застосунку Allplan.

Об'єктом дослідження є інформаційні технології додатків BIM системи Allplan.

Предметом дослідження є інформаційні технології підсистеми PytonParts в системі Allplan.

Для виконання мети були сформовані наступні задачі, що визначили структуру дослідження:

– проаналізувати існуючі рішення, що основані на подібних модулях;
– розробити алгоритмічну базу пошуку точок та взаємодії користувача з PythonParts модулем;
– реалізувати PythonPart модуль для створення армування.

Вхідною інформацією є наступні дані:

– вершини кожного шару стіни зі всіма вирізами;
– вершини кожного шару кожного з вирізів.

Результуючою інформіцією є наступні дані:

– відображення або створення моделі металоконструкції, створеної на основі заданих параметрів.

Функціонал завдання відповідає технічному завданню, яке вказане у додатку В.

В якості інструменту для створення PythonPart модуля було використано мову програмування Python версії 3.8.4, так як вона дає змогу, підключивши .pyd (C++) модулі, які постачаються разом з застосунком Allplan, реалізувати алгоритм у найкоротший час без втрати швидкодії застосунка.

1. ОГЛЯД СИСТЕМ У СФЕРІ BIM ПРОЕКТУВАННЯ АРХІТЕКТУРНИХ СПОРУД

1.1 Програмний комплекс для автоматизованого проектування Revit

Revit – це продукт Autodesk, котрий використовується для створення та роботи з інформаційним моделюванням будівель (BIM). Він містить складну геометрію та атрибутивну інформацію, які описують будівлю та її конструкцію. Наприклад, геометрія містить складні данні для таких об'єктів, як стіни та двері, і дозволяє інженерам класифікувати ці об'єкти за допомогою інформації з реального життя[1].

Інформаційне моделювання будівлі представляє собою трьохвимірний цифровий комп'ютерний проект для реальної структури. Частини геометрії інформаційних моделей будівель містять також фізичні та логічні характеристики структури міського середовища. Геометрія покаже стілець, але атрибути будуть містити модель, розмір та ідентифікаційний номер[1].

Переваги:

- Реалістична візуалізація;
- Можливість виміряти вплив будівлі на навколишнє середовище;
- Швидкий доступ до даних по документації проекту;
- Велика початкова бібліотека елементів;

Недоліки:

- Зосередження на структурному дизайні, що не дає можливості достатньо поглибитися у проектуванні будівлі;
- Редагування тільки у звичайній ізометричній перспективі, що ускладнює роботу для користувача, який вже знайомий з різними способами відображення моделей у інших системах;
- Невеликий розмір ком'юніті, зв'язаний з невисокою популярністю;

1.2 Програмне забезпечення для інформаційного моделювання будівель Tekla

Tekla Structures – це BIM програмне забезпечення, що дозволяє створювати, комбінувати, управляти та поширювати багатоматеріальні 3D моделі, запаковані з цінною конструкторською інформацією. Tekla Structures можна використовувати в усіх типах проектів, починаючи з будівель та концептуального планування інфраструктури до виготовлення, конструювання та підтримки, для дизайну, деталізації та управління інформацією про проекти[2].

Переваги:

- Широкий вибір елементів, доступних в початковій бібліотеці;
- Креслення можуть бути збережені у форматі .dwg, що дає змогу маніпулювати їми в AutoCAD;
- Центральна база даних, що дає впевненість про зкординованість креслень та звітів з моделлю
- Якісна підтримка користувача

Недоліки:

- Складне програмне забезпечення з дуже перенавантаженим інтерфейсом;
- Недостача в інструментарії моделювання нестандартних об'єктів;
- Документація повністю написана у текстовому форматі, що дуже ускладнює процес вивчення системи, особливо без досвіду з програмами такого типу. Скоріше за все доведетсья звернутися до професіональних курсів або ментора;
- ПЗ дорожче, ніж у конкурентів

1.3 Порівняння застосунків

Існує багато різноманітних рішень. Такі, як, наприклад, AutoCAD, мають зовсім іншу систему роботи з будівельними спорудами, а інші, як SketchUp мають набагато меншу функціональність.

Якщо ж брати до уваги системи, освітлені у даному розділі, то перша система, тобто Revit, взагалі не має функціоналу армування, який потребується модифікувати.

Друга система, Tekla, має широкий функціонал можливостей і, на відміну від першой системи, може впоратися з завданням армування. Але вона на порядок складніша як для загального освоєння, так і зокрема для армування. Також системі бракує можливості додавати вузькоспеціалізовані елементи у короткий термін, такі як армування саме стін з вирізами, які виходять за рамки звичайного армування.

Тож Allplan має досить гнучку систему створення моделей армутури, яка повинна покривати як прості, так і складніші потреби користувача. Також система PythonParts дозволяє створити елементи будь-якої складності засобами розробника, шо дозволяє досить швидко модернізувати та підлаштовувати систему під потреби користувачів.

1.4 Висновки до розділу

У даному розділі зібрано інформацію про альтернативні конкуруючі застосунки Revit та Tekla, яка показала актуальність розробки модуля PythonParts.

2. ЗАСОБИ РОЗРОБКИ

2.1 Вибір технологій та їх обґрунтування

Так як завдання передбачає створення PythonPart модуля за допомогою влаштованої в Allplan системи, то основною для розробки є мова Python. Також використовуються реалізовані за допомогою мови C++ pyd модулі для роботи з 3D графікою та інтеракцією з Allplan елементами. Так як в системі PythonParts використовується дебагер PTVSD, який наразі вважається застарілим, то єдиним доступним середовищем розробки, яке може працювати з цим дебагером, можна вважати Visual Studio 2017 року або раніше. Для впровадження інтерфейсу користувача буде використовуватися система з інтерактивними палетками, також вбудована в Allplan.

2.2 Мова програмування Python

Python – це універсальна інтерпретована, об'єктно-орієнтована високорівнева мова програмування сценаріїв із динамічною 18 семантикою. Розвинені вбудовані структури даних у поєднанні з динамічною типізацією та динамічним зв'язуванням роблять її дуже привабливою для швидкої розробки застосувань, а також для використання як скриптової або мови, що "склеює" разом наявні компоненти. Мова Python є простою, має легкий у вивченні синтаксис, забезпечує високу читабельність і через те зменшує загальну вартість експлуатації програм. Python підтримує модулі та пакети, які сприяють мобільності програм і повторному використанню коду. Інтерпретатор Python та розширена стандартна бібліотека доступна як у вихідному коді, так і у двійковому форматі, причому безкоштовно і для всіх основних платформ. Його можна вільно поширювати та навіть убудовувати у власні застосування[3].

2.3 PYD модулі для Python

Існує три типи файлів для зберігання байт коду Python – з розширеннями .pyc, .pyd та .pyo. PYC та PYO використовуватися не будуть, але важливо відмітити модулі з розширенням .pyd

Тип файлу .pyd, на відміну від інших типів, призначений тільки для сімейства OS Windows. Таким чином він може зустрітися на дистрибутивах встановлених програм у персональних та корпоративних версіях Windows 10, 8, 7 та ін.

В екосистемі Windows файл .pyd представляє собою файл бібліотеки, котрий містить код Python, який може бути викликаний та використовуватися іншими додатками Python. Щоб зробити цю бібліотеку доступною для інших програм Python, вона упакована у виді динамічно підключаємої бібліотеки.

Динамічно підключаємі бібліотеки (DLL, dynamic link library) – це бібліотеки зкомпільованого коду Windows, котрі підключаються у програму, що їх викликає, під час виконання. Основною перевагою таких бібліотек є те, що вони полегшують повторне використання коду, дозволюють будувати модульні архітектури та прискорюють запуск програм. В результаті бібліотеки DLL забезпечують велику функціональність в ОС Windows.

Файли .pyd – це dll, але є декілька відмінностей. Якщо є файл з ім'ям hello.pyd, він повинен містити функцію inithello() як точку входу. Потім можна написати у своєму модулі Python "import hello" і інтерпретатор буде шукати hello.pyd (а також hello.py, hello.pyc), та якщо він знайде його, спробує викликати inithello() для його (.pyd файла) ініціалізації[4].

Конкретно у випадку з Allplan модулями – вони створені на основі C++ та мають у собі широкий спектр можливостей. В основному в цій роботі використовується логіка для інтеракції з об'єктами (стінами та вирізами) у системі та загальна 3D логіка, яка дозволяє створити модель каркасу арматури, відштовхуючись від параметрів стіни.

2.4 Середа розробки Visual Studio 2017

Microsoft Visual Studio – це набір засобів для створення ПЗ зі стадії планування до стадії UI (user interface) тестування, кодингу, тестування, дебагу, алалізу якості та продуктивності коду, розгортання для користувача та збору телеметрії щодо використання. Ці засоби створені для роботи разом настільки плавно, наскільки це можливо та всі вони представлені Visual Studio IDE (integrated development environment)[5].

Це середовище дозволяє працювати з .pyd файлами з використанням помічника IntelliSense, який доступний як у Visual Studio, так і у звичайному редакторі Visual Studio Code.

Visual Studio IntelliSense представлений для різних мов програмування та різних форматів файлів (від звичайних логів до JSON/XML) та підтримує основаного на реальних проектах помічника у завершенні коду та може бути зконфігурованим для надання більшої варіативності у підказках[6].

Цей засіб допомагає без зайвих звернень до документації PythonParts використовувати влаштовані .pyd модулі, отримуючи інформацію про реалізацію цих файлів прямо у середовищі.

2.5 Дебагер PTVSD

The Pyton Visual Studio Debugger (PTVSD) – механізм, котрий реалізовує Visual Studio протокол дебагу та використовується як знаряддя дебагу у Visual Studio та Visual Studio Code[7].

Хоча дебагер PTVSD, створений корпорацією Microsoft, вважається застарілим, але свою функцію для PythonParts він виконує. За домопогою цього засобу можна динамічно відслідковувати стан об'єктів, їх змінні у потрібний момент виконання програми. Також він дозволяє динамічно створювати нові об'єкти, коли програма знаходиться в очікуванні на точкі зупину, що спрощує занурення в .pyd файли та вже готову реалізацію модулів в системі PythonParts.

2.6 Вбудована в Allplan система створення PythonParts

Для створення палетки задання параметрів використовується система, суміжна з файлами типу .xml.

The Extensible Markup Language (XML) – це підмножина SGML. Завдання XML – зробити можливим надання, отримання та обробку загального SGML файлу у Web середовищі шляхом, схожим з HTML. XML був розроблений для простоти реалізації та сумісності з SGML та HTML[8].

Загальний синтаксис XML файлу наступний:

```
<root>
  <child>
    <subchild>.....</subchild>
  </child>
</root>
```

Він має вкладену структуру, де root – корінь, від якого відходять інші елементи, а ті, в свою чергу, також можуть мати нащадків.

У ситсемі Allplan використовуються модифікаовані XML файли з розширеннями .pyp, .pal, які залежать від типу потрібного елементу (розширювач, інтерактор та ін.). За синтаксисом вони, як і звичайні XML, мають містити головні елементи, від яких створюються дочірні.

Мінімальне визначення pyp (або pal) файлу повинно містити хоча б один <Element> та хоча б один <Script> вузол. Підвузли у <Script> визначають посилання на відповідний Python скрипт всередині <Name> вузла. <Title> вузол визначає заголовок для PythonParts палетки. <Version> вузол визначає версію PythonPart модуля. І останній в <Script> – <ReadLastInput> вузол визначає чи повинна палетка відкриватися з останньої збереженої точки (якщо така є)[9].

Вузол <Element> зазвичай містить у собі хоча б один вузол <Page>.

Для створення сторінки всередині параметричної палетки в Allplan, у pyd файлі повинен бути використаний вузол <Page>. Всі параметри палетки для користувача повинні бути дочірнімі по відношенню до <Page>[9].

Що стосується самих параметрів, то вони мають деяку схожість з HTML документом, який представляє собою розмітку для веб-сторінки.

Одними з основних елементів можуть еспандери у яких зберігаються параметри та самі параметри різних типів.

Тут введена реалізація як звичайних типів, такие як числове значчення, текст, логічні значення (чекбокси та радіобаттони), але дещо розширена. Так, наприклад число може виступати величиною кута, або довжини, зважаючи на потреби. Також є багато передналаштованих списків, наприклад відступ від бетонного краю, значення якого можуть задаватися кожним користувачем індивідуально.

2.7 Висновки до розділу

Проведено аналіз засобів розробки, який показав доцільність використання Visual Studio 2017, влаштованого в неї дебагера PTVSD та мови програмування Python з .pyd модулями

3. АРХІТЕКТУРА ПРОГРАМНОЇ СИСТЕМИ

3.1. Алгоритм армування

Загальний алгоритм інтеракцій користувача зображений на діаграмі 3.1.1. На цій діаграмі можна спостерігати, що після певних дій корисувача модуль починає створення або передпоказу армування, або самого армування в системі для його подальшого використання.

Сам алгоритм армування у загальному вигляді приведений на діаграмі 3.1.2. Оркестратори у випадку армування – класи, які містять в собі крайні точки стіни або вирізу та всі їх нормалі. Також вони мають методи для обробки цих даних. Відштовхуючись від наведених у пункті 3.1 параметрів, програма створює армування потрібних типів.

Як вже було згадано вище, програма працює с армуваннями трьох типів – периметрально, по площі на вирізів. У діаграмах 3.1.3, 3.1.4 та 3.1.5 наведені більш поглиблені алгоритми реалізації модуля PythonPart.

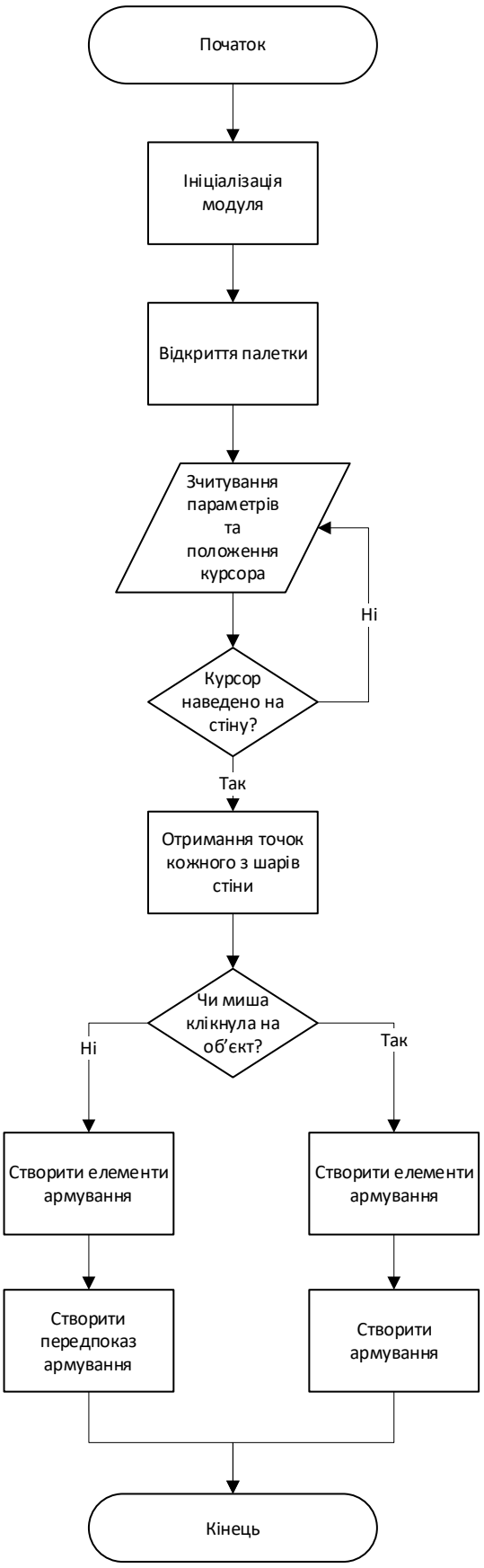

Рисунок 3.1.1 – Діаграма роботи програми

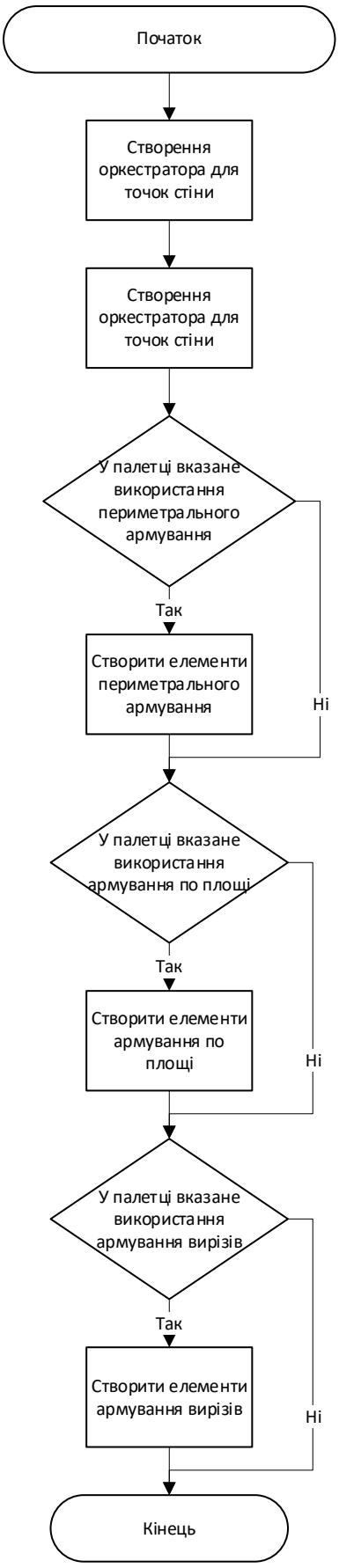

Рисунок 3.1.2 – Діаграма алгоритму армування стіни

Рисунок 3.1.3 – Діаграма алгоритму периметрального армування

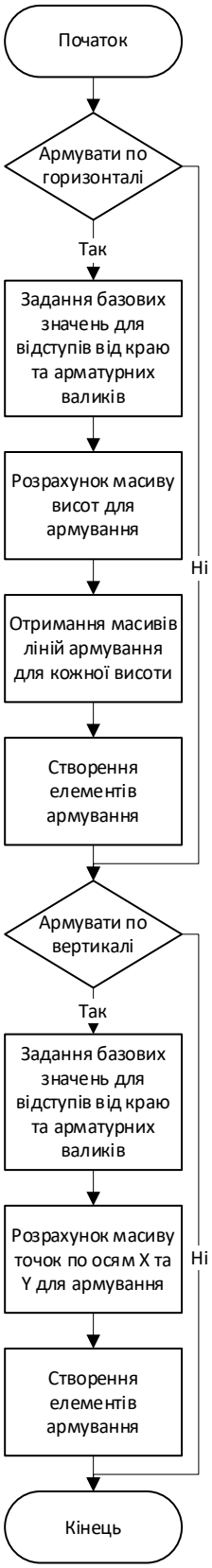

Рисунок 3.1.4 – Діаграма алгоритму армування по площі

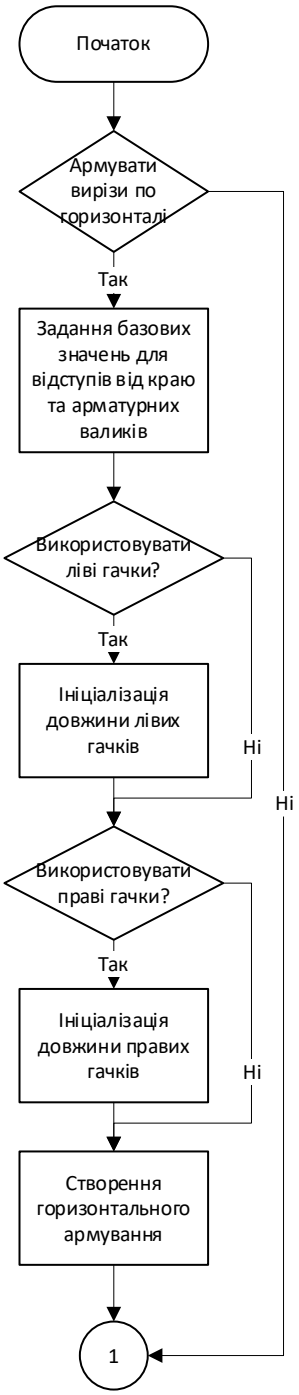

Рисунок 3.2.5 – Діаграма алгоритму армування вирізів

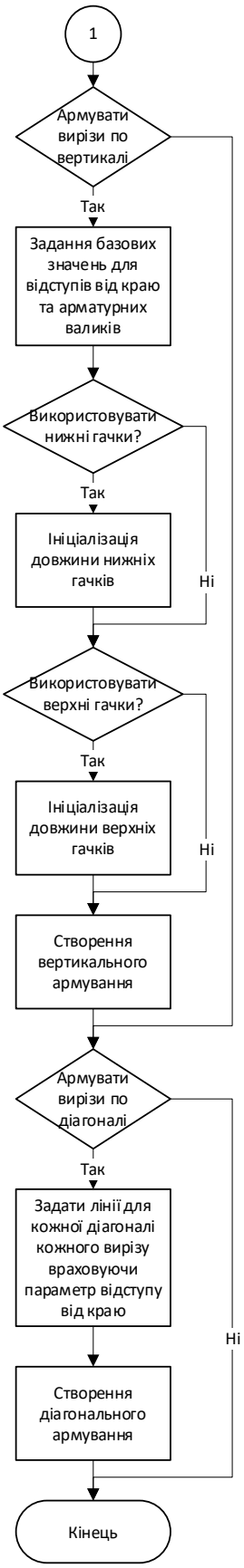

Рисунок 3.2.5 – Діаграма алгоритму армування вирізів (продовження)

На цих діаграмах зображена логіка створення армування для різних типів. Загалом на кожному типі армування використовуються схожі принципи, так як зчитування параметрів та задання базових значень та станів об'єктів. Хоча ця логіка дійсно повторюється не один раз, все одно не можна використати принцип DRY (don't repeat youself) на даних алгоритмах, бо вони беруть параметри з палетки, яка задається користувачем. Також, з першого погляду може здатися незрозумілим, чому при використанні гачків не створюється об'єкт для їх параметрів, а вони відразу ініцілізуються. Це тому, що система Allplan заздалегідь налаштована на тип гачків, що не мають довжини – це означає, що гачки без довжини просто не будуть враховуватися при армуванні.

3.2. Алгоритм пошуку точок стіни

Так, як сам Python являє собою мову програмування, яка підтримує ООП, то було прийняте рішення створити не тільки сам інтерактор для обробки вводу користувача, а і окремий клас для стін та вирізів, що основується на отриманих вершинах. Цей клас називається ModelsOrchestrator. Також весь модуль повинен відповідати сигнатурі модуля – інтерактора та реалізовувати деякі методи, включаючи create_interactor(coord_input, pyp_path, str_table_service) у якому ініціалізується сам інтерактор. Загальна модель класів приведена на рис. 3.2.1.

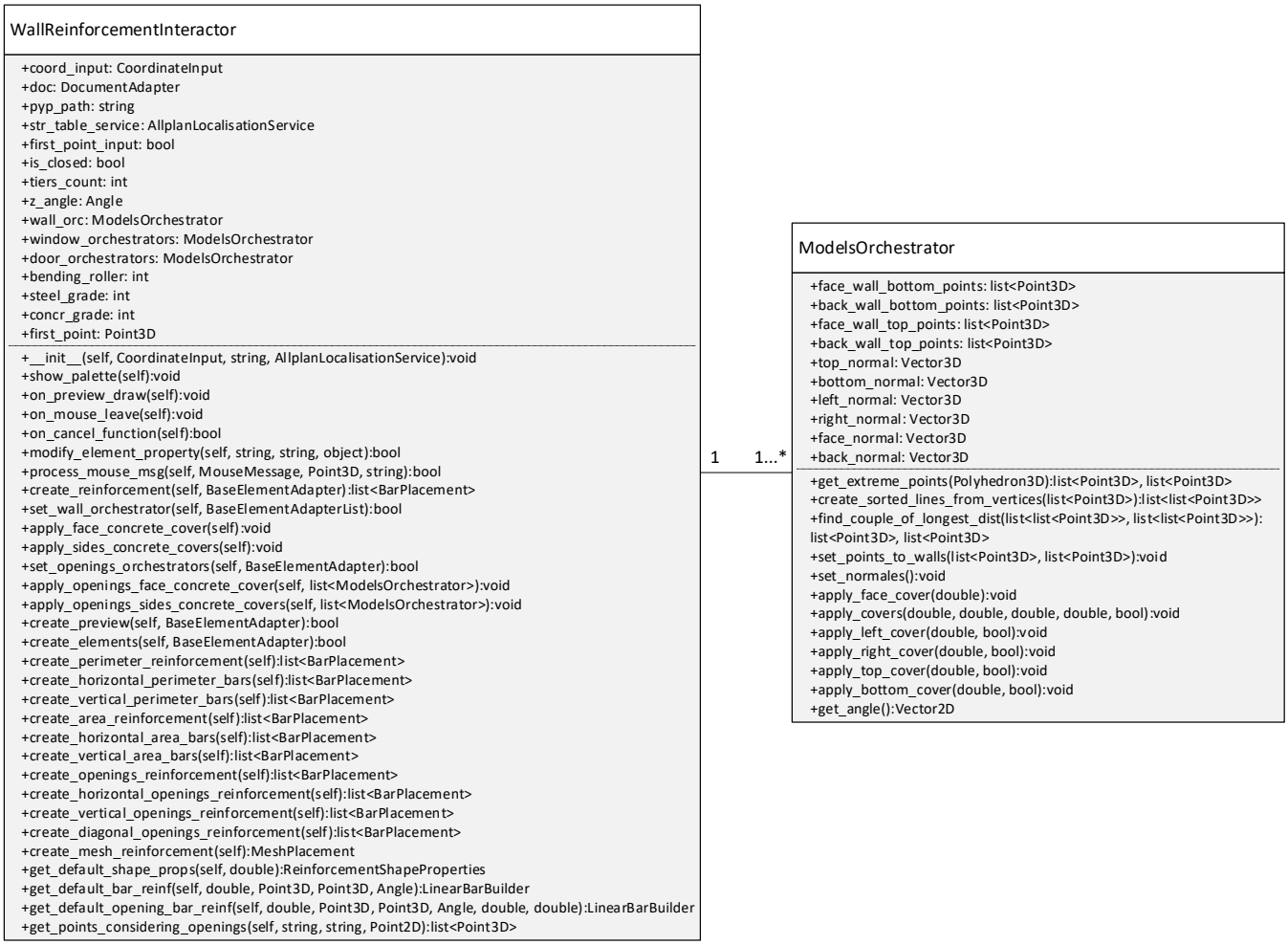

Рисунок 3.2.1 – Діаграма класів модуля армування стін

Логіку інтерактора загалом описано у пункті 3.2 цього розділу, тому у цьому пункті буде пояснення щодо алгоритму пошуку точок та нормалей для об'єктів, який реалізовано в ModelsOrchestrator.

Цей клас приймае два об'єкти типу Polyhedron3D, який реалізовано у Geometry.pyd модулі. Передбачається, що цими об'єктами будуть перший та останній шари стіни або вирізу. Саме два шари потрібні для того, щоб скординувати крайні точки, якщо у стіни декілька шарів. Алгоритм також буде працювати, якщо у стіни тільки один шар і в конструктор ModelsOrchestrator передається один і той самий об'єкт або два однакових об'єкта.

Цей клас працює покроково, тому його роботу можна розділити на декілька етапів:

– Отримання двох об'єктів типу Polyhedron3D;
– Оголошення початкових значень;

– Отримання крайніх точок кожного шару;
– Створення сортованих масивів точок для кожного шару, які містять в собі інформацію про крайні лінії для армування стіни;
– Отримання пар ліній, які знаходяться на найбільшій відстані одне від одного. Окремо для верхніх та нижніх ліній;
– Встановлення значень ліній у оголошені змінні;
– Встановлення нормалей, опираючись на створені лінії;

Після всії цих маніпуляцій ми отримуємо об'єкт, що містить у собі значення переднього та задного шарів стіни/вирізу, всі нормалі до поверхностей та метод get_angle(), котрий обраховує кут повороту стіни у загальній системі координат.

Деякі з цих етапів схематично зображено на діаграмах 3.2.2, 3.2.3, 3.2.4 та 3.2.5.

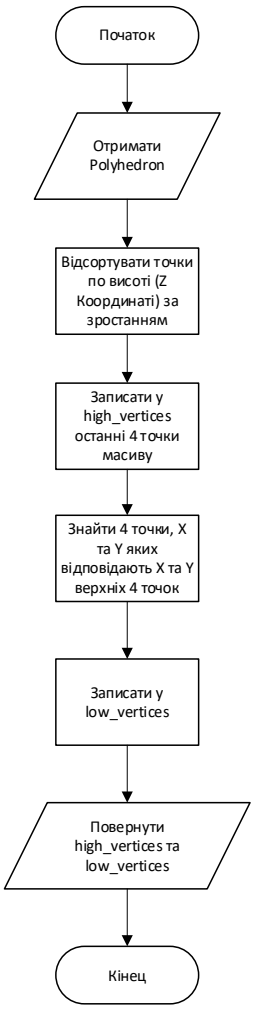

Рисунок 3.2.2 – Діаграма алгоритму отримання крайніх точок кожного шару

Рисунок 3.2.3 – Діаграма алгоритму створення сортованих масивів точок для кожного шару, які містять в собі інформацію про крайні лінії для армування стіни

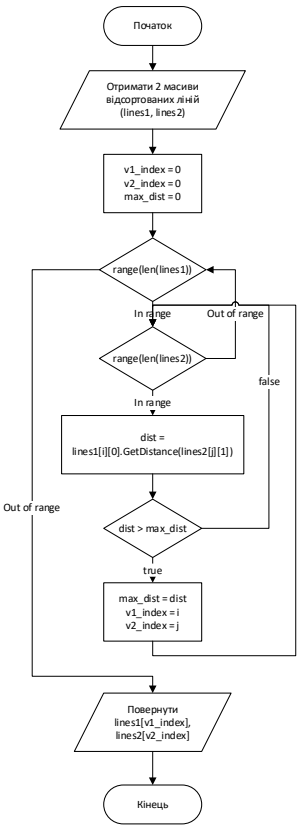

Рисунок 3.3.4 – Діаграма алгоритму отримання пар ліній, які знаходяться на найбільшій відстані одне від одного

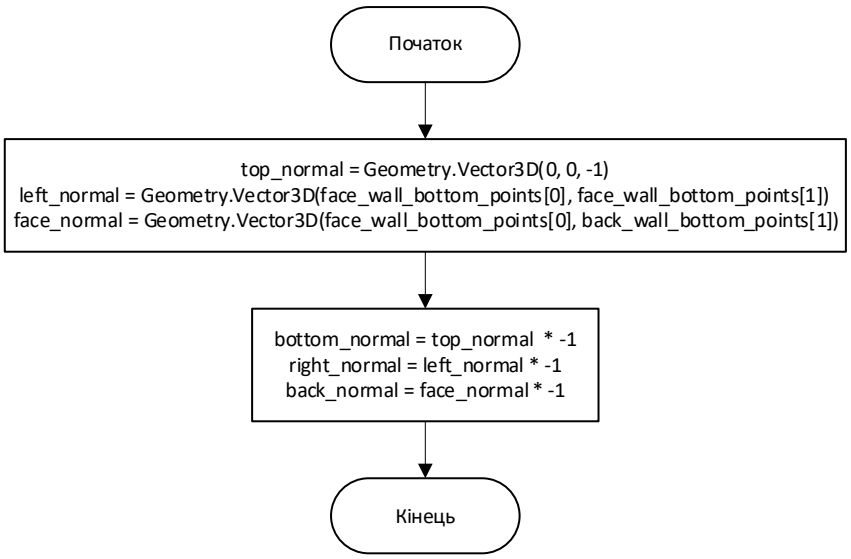

Рисунок 3.3.5 – Діаграма алгоритму встановлення нормалей

3.3. Структура pal модуля

PAL модуль, як вже згадувалося, належить сімейству XML документів, тому все, що треба для функціонування програми це налаштувати шлях до .py модуля та розмапити параметри по необхідним місцям.

Для реалізації цього процесу було створено .pal модуль, який автоматично завантажується при виклику користувачем необхідного модуля.

У якості іконки для модуля було взято просте зображення армованої п'ятикутної стіни – рис. 3.1.1.

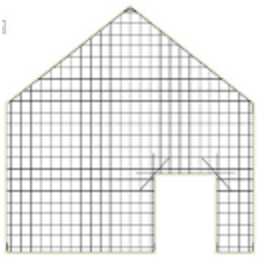

Рисунок 3.1.1 – Іконка модуля

Так, як у завданні досить багато параметрів для різних типів армування, то .pal модуль розбитий на логічні рівні, що дозволяє спростити навігацію по палетці як користувачеві, так і розробнику по самому файлу. Розбиття було виконано за допомогою еспандерів, які у самій програмі виглядають як на рис. 3.1.2.

Рисунок 3.1.2 – Рисунок еспандера для загальних параметрів армування

Також для користувача була додана можливість обирати потрібний тип армування за допомогою чекбоксів, що дає змогу відсікти непотрібне армування, наприклад периметральні арматурини. Сюди ж для спрощення була додана можливість обирати який саме шар (передній чи задній) буде створено, також за допомогою чекбоксів – рис. 3.1.3.

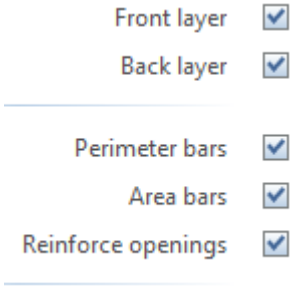

Рисунок 3.1.3 – Вибір шарів та типу армування

Так, як деякі типи армування мають загальні властивості, то такі параметри, які бетонний відступ від краю було винесено у загальну зону, яка не відноситься до окремого армування. Ці параметри зображені на рис. 3.1.2 та рис. 3.1.4.

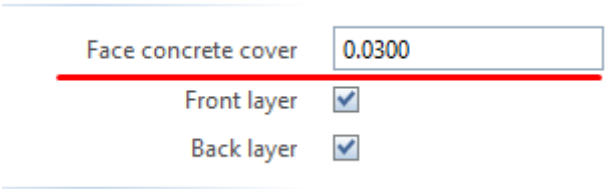

Рисунок 3.1.4 – Окремий параметр для відступу від лицевої сторони

Загалом кожне армування, а саме – периметральне (горизонтальне на вертикальне), по площі (горизонтальне та вертикальне) та вирізів (горизонтальне,

вертикальне та діагональне), обов'язково має свій параметр діаметру арматури. На рисунках 3.1.5, 3.1.6 та 3.1.7 можна побачити параметри для кожного типу армування.

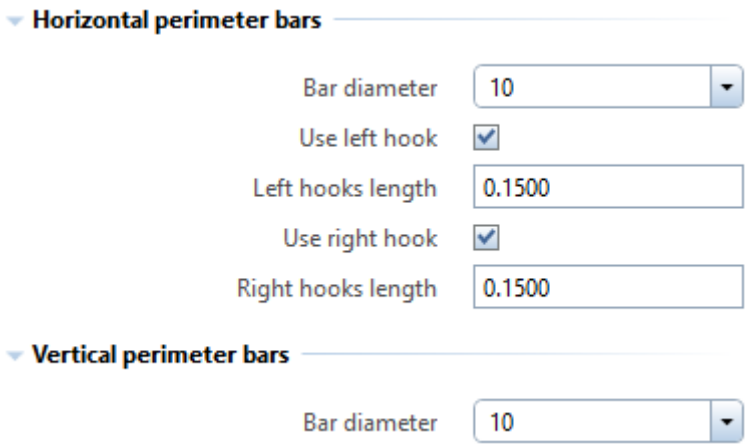

Рисунок 3.1.5 – Параметри для периметрального армування

Рисунок 3.1.6 – Параметри для армування по площі

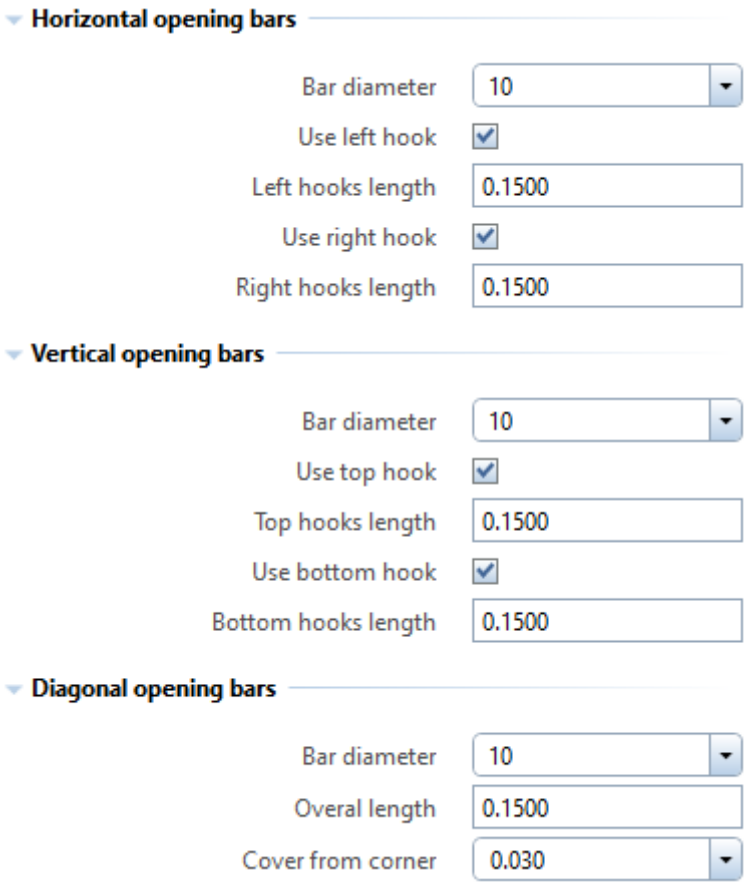

Рисунок 3.1.7 – Параметри для армування вирізів

Також важливо відмітити, що армування вирізів має окремі значення параметрів для відступу, так як в них рахується відступ не від краю стіни, а від краю самих вирізів. На рис. 3.1.7 та 3.1.8 зображені відступи від краю для діагонального та від ребер для периметрального армувань.

Рисунок 3.1.8 – Параметри відступу для діагонального армування

Згідно з завдання, горизонтальне периметральне армування та пряме (горизонтальне та вертикальне) армування вирізів мають можливість створення гачків для арматури та вказання їх довжини, що зображено на рисунках 3.1.5 та 3.1.7.

Для армування по площі необхідно задати значення відступу між самими арматуринами для розрахунку їх загальної форми, відштовхуючись від довжини стіни, тому, як показано на рис. 3.1.6, палетка має параметри для задання проміжку між окремими елементами.

Останнім параметром є довжина діагонального армування вирізів, яка зображена на рис. 3.1.7 у останньому еспандері.

3.4. Висновки до розділу

У цьому розділі описані основні алгоритми для інтерактора та оркестратора точок, що виконують задачу армування стін з урахуванням параметрів користувача.

4. РОБОТА КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

4.1. Робота з палеткою параметрів

Для початку користувач повинен зайти в папку з потрибним PythonPart модулем, що можна зробити через вкладку Library (див. рис. 4.1.1).

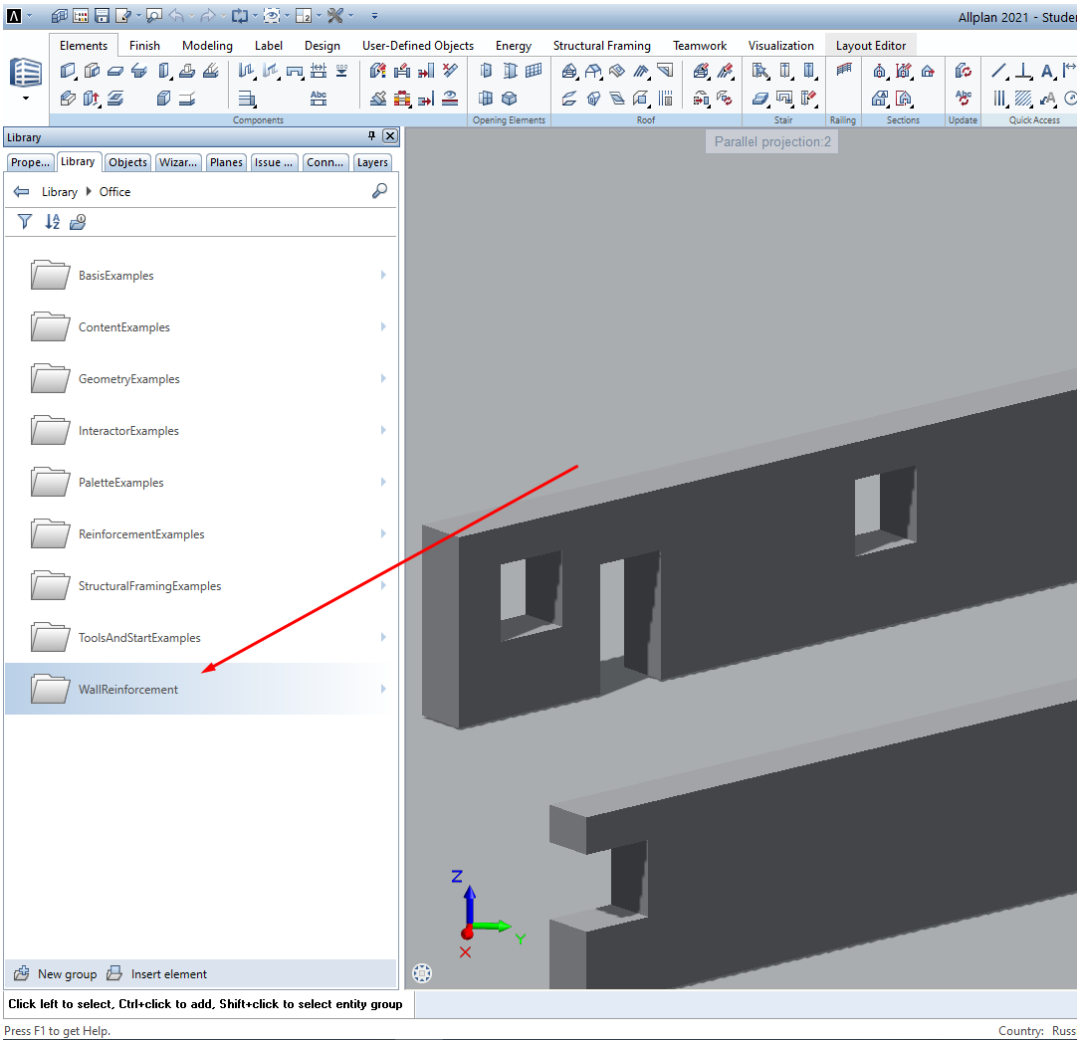

Рисунок 4.1.1 – Інтерфейс Allplan з відкритою вкладкою Library

При вході, як і в інших папках, у яких знаходяться .pyd або .pal файли, користувач бачить прев'ю форми, задаваємої PythonPart модулем, налаштування шрифтів та самі модулі, доступні у цій папці (рис. 4.1.2).

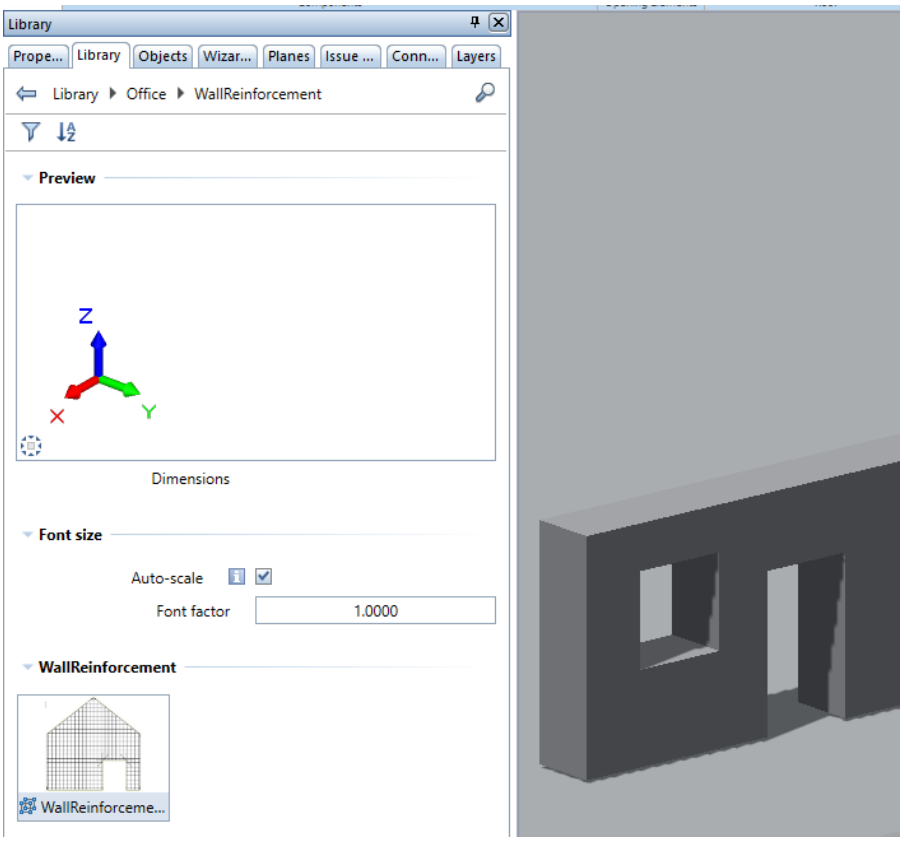

Рисунок 4.1.2 – Інтерфейс відкритої папки з модулем

Також в системі доступні різні види проекцій, що надає гнучкості у роботі з Allplan. У нашому разі відкрито 2 типи проекцій: 3D та 2D з видом згори (рис. 4.1.3).

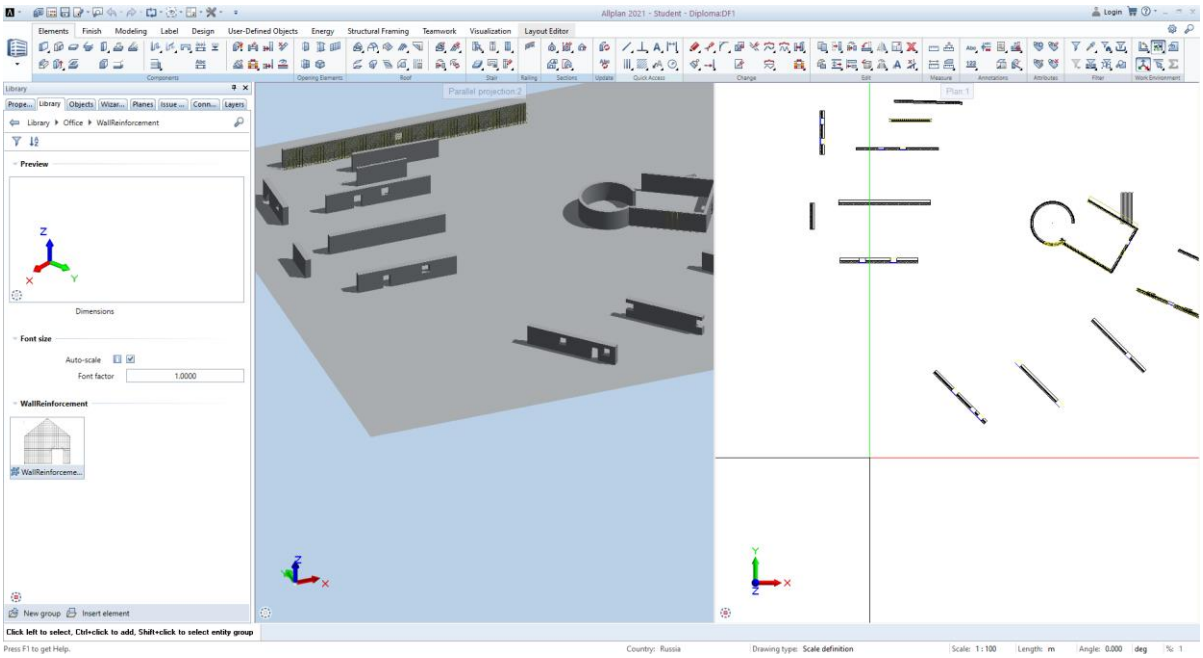

Рисунок 4.1.3 – Повний інтерфейс Allplan з двомі вікнами проекцій.

При необхідності користувач може сам налаштувати потрібне розташування вікон та їх кількості при зміні параметрів у табі Window згори програми (рис. 4.1.4).

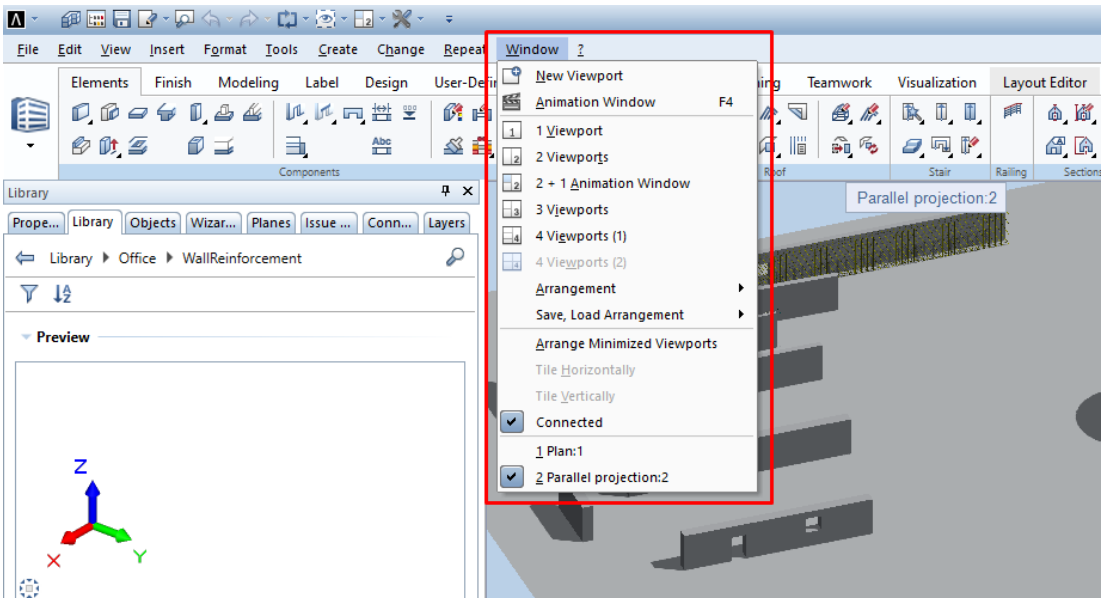

Рисунок 4.1.4 – Повний інтерфейс Allplan з двомі вікнами проекцій.

Для початку роботи з модулем – його потрібно перетягнути на одне з вікон проекту. Після цього відкриваєтья палетка, яка містить у собі параметри для армування стіни. Функціональність майже всіх параметрів описана у першому пункті третього розділу (3.1), за винятком вказання типу армування. Наразі створено тільки армування за допомогою стержнів арматури (Bar), але при необхідності можна додати Mesh, тобто арматурні сітки. Вся палетка зображена на рисунках 4.1.5 та 4.1.6.

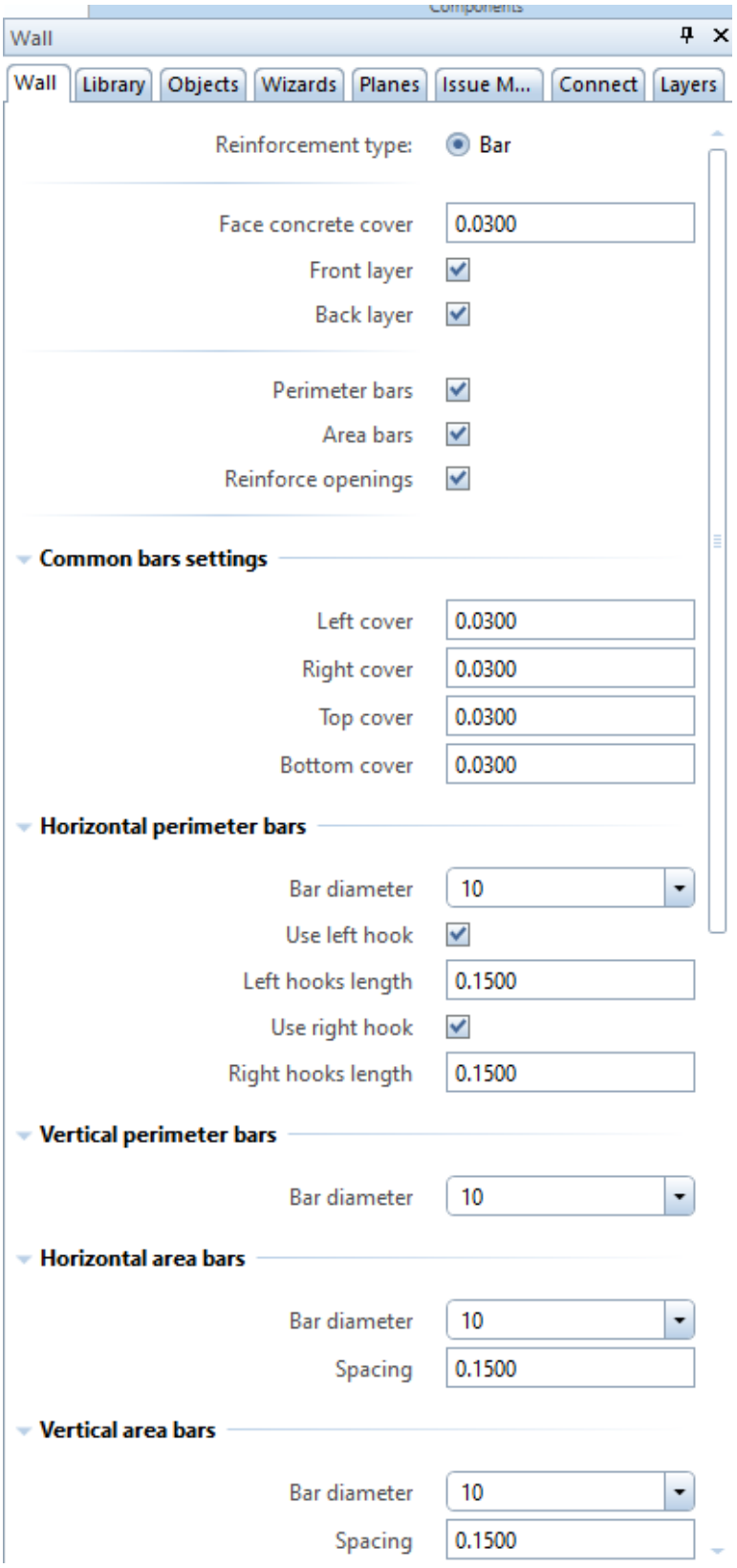

Рисунок 4.1.5 – Перша частина палетки

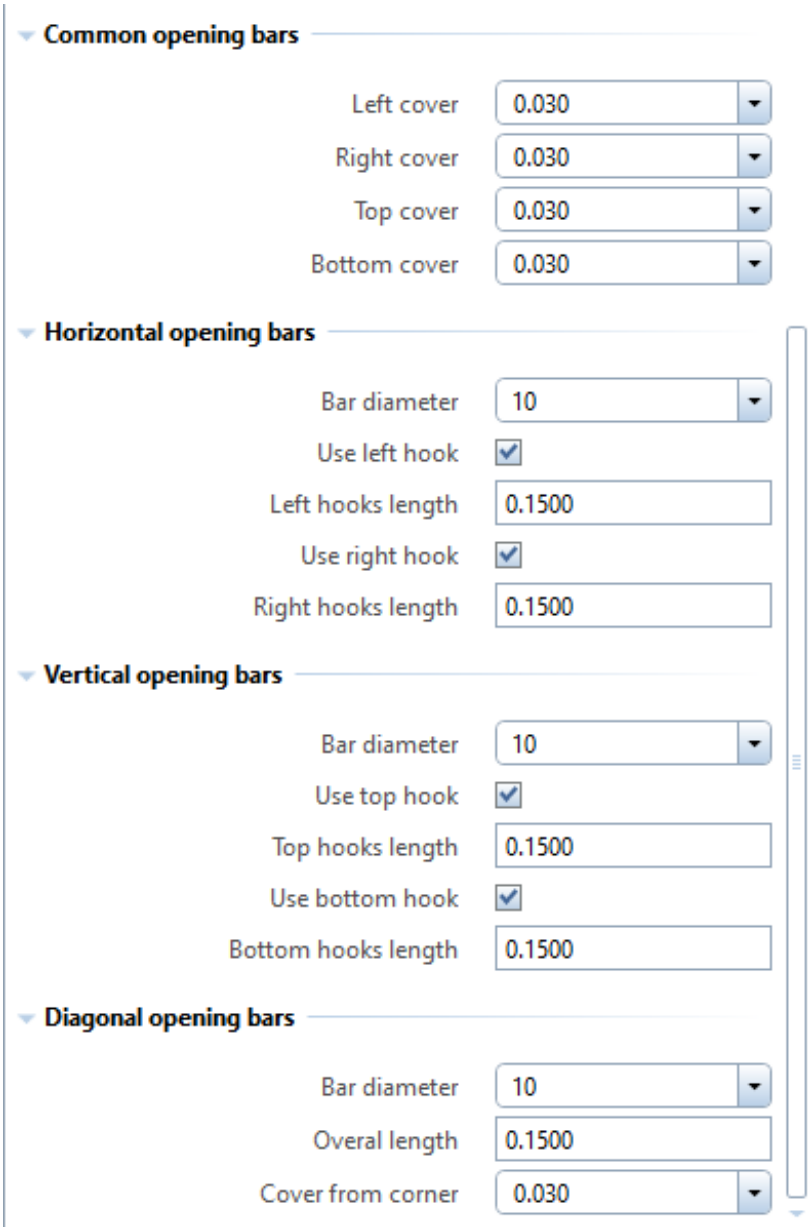

Рисунок 4.1.6 – Друга частина палетки

Тут користувач має задати потрібні йому параметри для необхідного армування. На деяких елементах налаштована видимість в залежності від стану чекбоксу, тобто при вимкненні всіх типів армувань користувач буде бачити тільки загальні налаштування відступу від краю (рис. 4.1.7). Також при вимкненні будь-якого з гачків, наприклад, на горизонтальному периметральному армуванні, зникає поле, де вказується довжина гачка (рис 4.1.8).

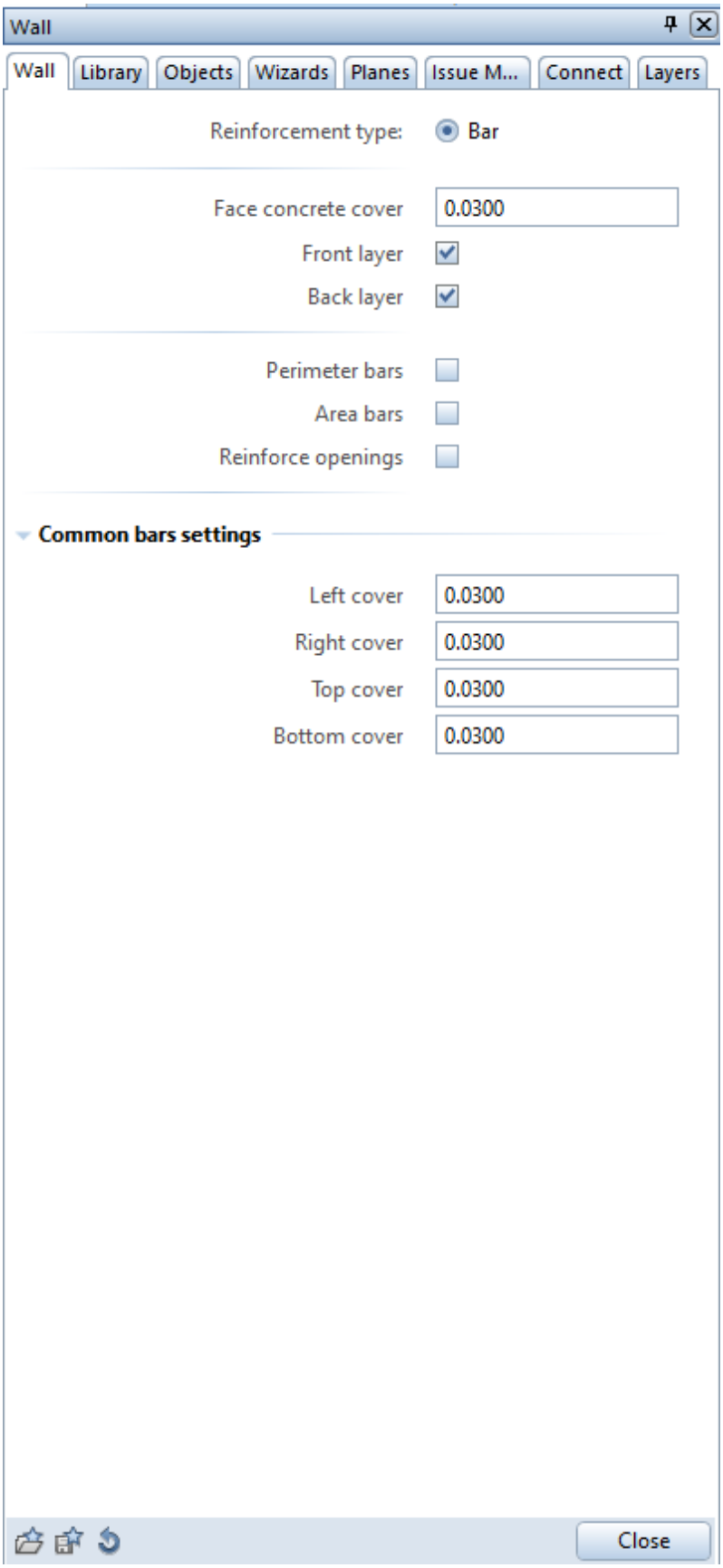

Рисунок 4.1.7 – Всі типи армування вимкнені

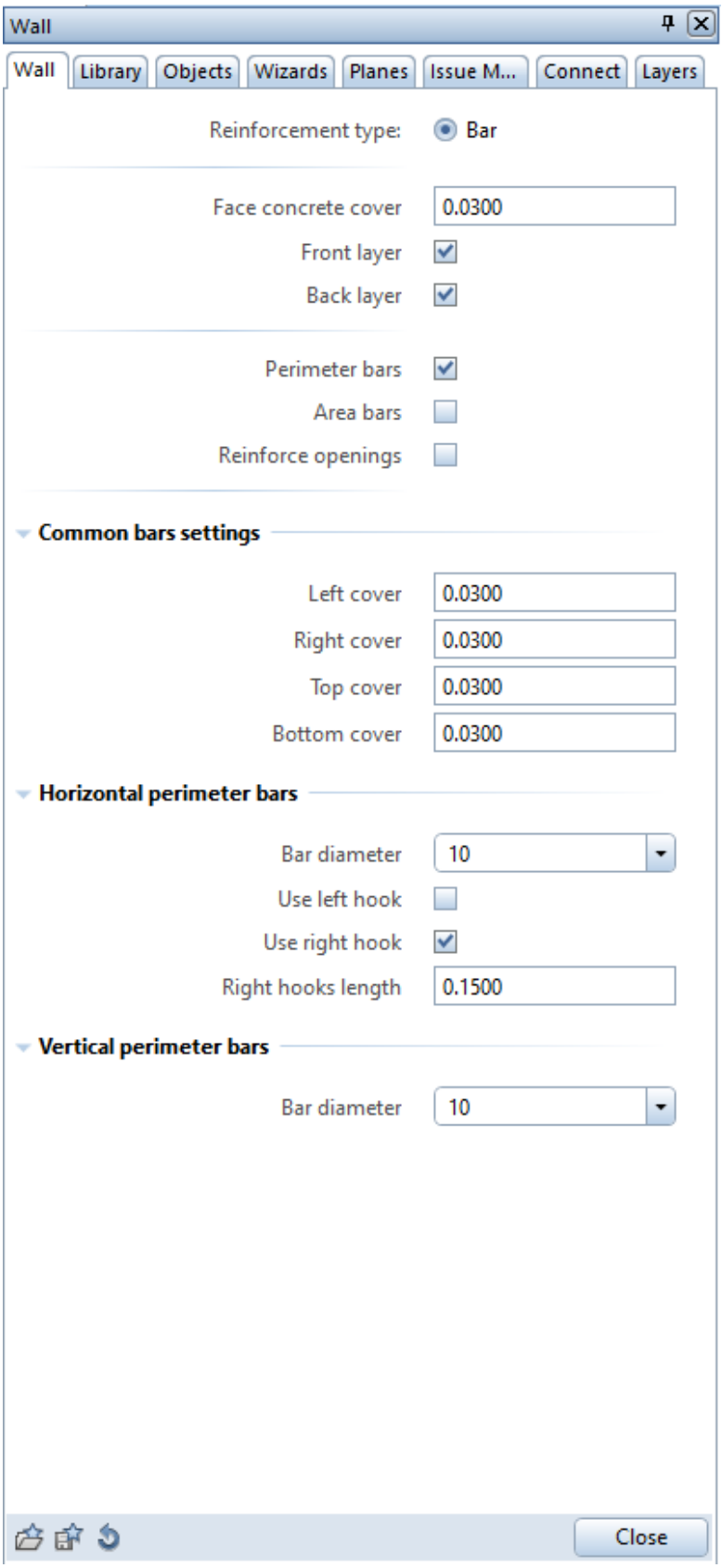

Рисунок 4.1.8 – Вимкнені ліві гачки у горизонтального армування

4.2. Робота з армуванням стіни у різних проекціях (2D, 3D)

На відміну від Revit, Allplan дозволяє нам працювати у різних проекціях, таких як 2D та 3D. Як вказано в попередньому пункті, армування буде демонструватися одразу у двох проекціях (рис. 4.2.1 та рис. 4.2.2). І, нехай, у користувача є можливість працювати з армуванням у будь-якій зручній проекції, відрисовка армування буде подаватися здебільшого у 3D просторі, коли інтеракція у 2D, що спростить розуміння застосунка. Також буде використовуватися тільки один передній шар, задля розбірливості форми армування. Важливо відмітити, що стіна має чотири шари, третій та четвертий шар видно на рис. 4.2.2, у якому мишкою обрано четвертий шар, а між ним та затекстурованим лежить третій.

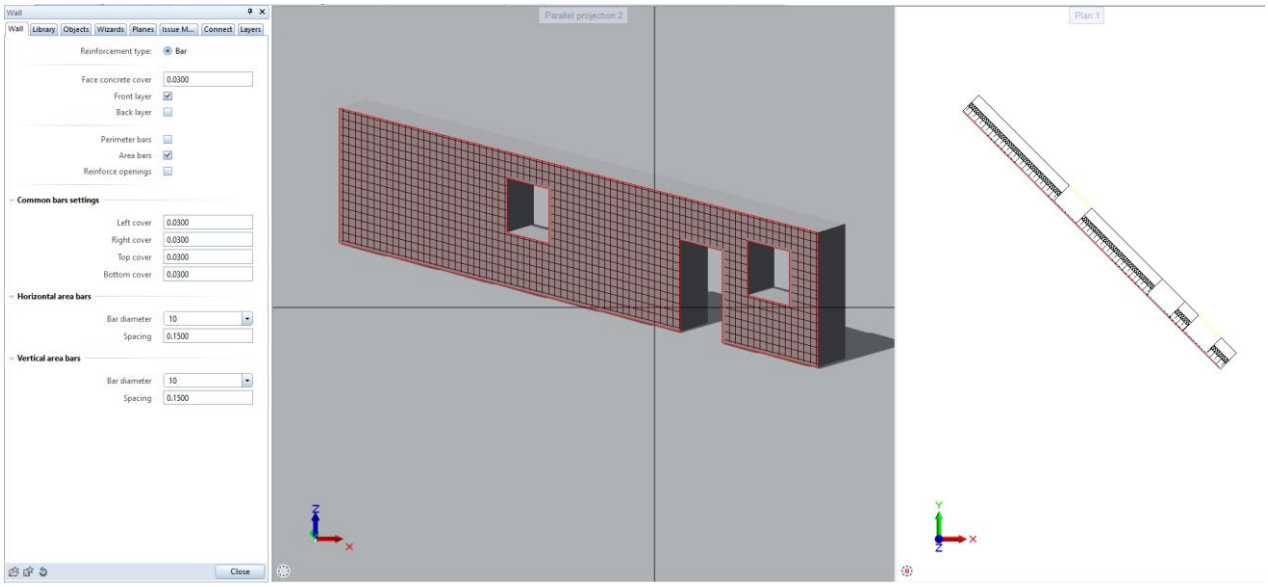

Рисунок 4.2.1 – Армування при інтеракції з 3D моделлю

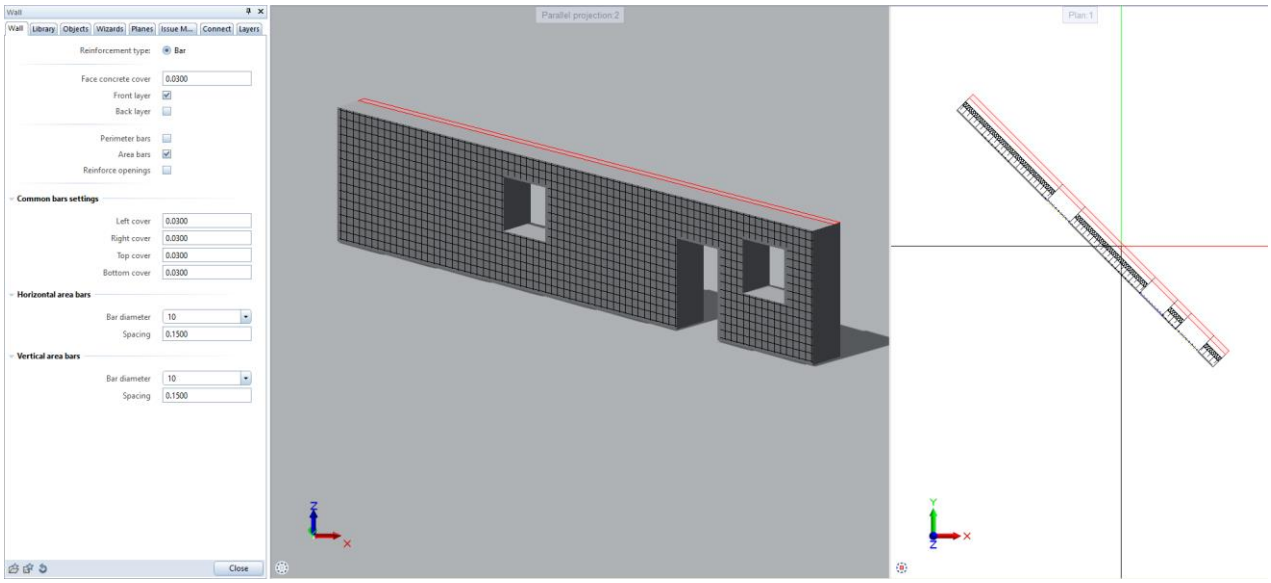

Рисунок 4.2.2 – Армування при інтеракції з 2D моделлю

Для того, щоб користувач побачив прев'ю армування потрібно просто встановити потрібні параметри та навести на стінку. Якщо вказані параметри не задовольнили потреби, то просто їх змінити та ще раз навести курсор на потрібну стіну. Не варто натискати на стіну при наведенні, якщо користувач не впевнений у фінальному результаті, так як при цій дії буде створене армування в базі даних (у проекті). У разі невдачі можна відключити вид стін, виділити арматуру та видалити її, почавши все заново (рис 4.2.3). Також модуль перестає працювати, створивши армування. При наведенні на стіну після натиснення на неї не буде з'являтися навіть прев'ю (рис. 4.2.4). Якщо потрібно створити ще одне армування стіни, то треба закрити палетку та відкрити нову.

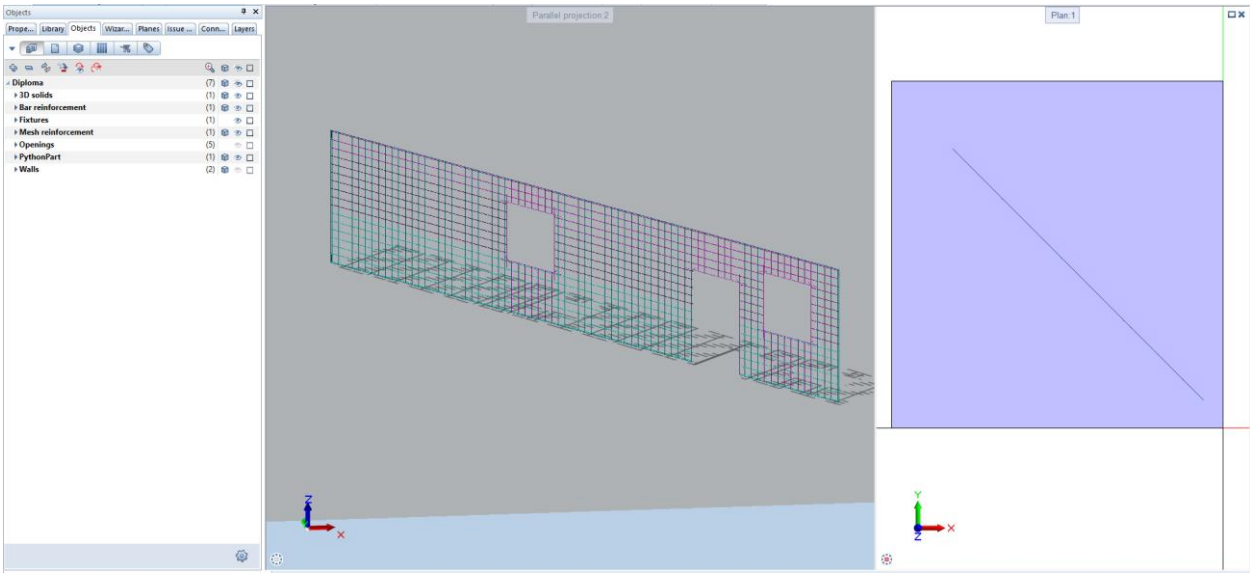

Рисунок 4.2.3 – Створена арматура та виділення армування при відключеному відображенні стін

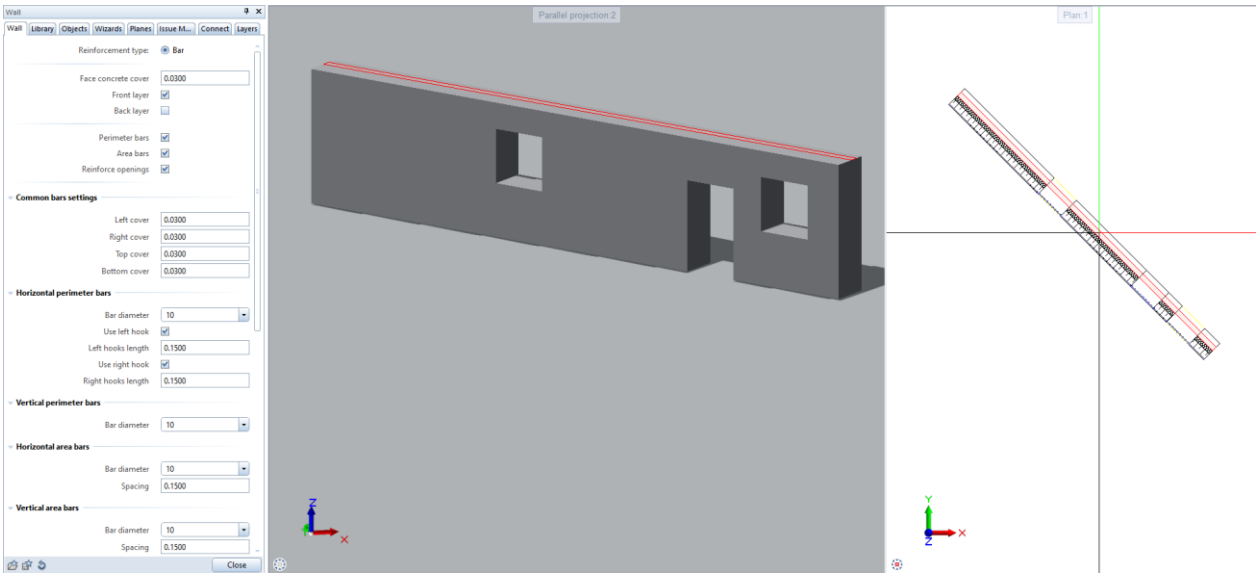

Рисунок 4.2.4 – Армування не створюється при повторній інтеракції

Введемо завеликі значення відступів від країв для на наочності. На рис зображене прев'ю армування при заданих відступах (зліва – 1м, зправа – 0.2м, згори – 0.3м та знизу – 0.4м). На рисунках 5.2.5 та 5.2.6 зображені армування тільки по периметру та всі типи армування відповідно.

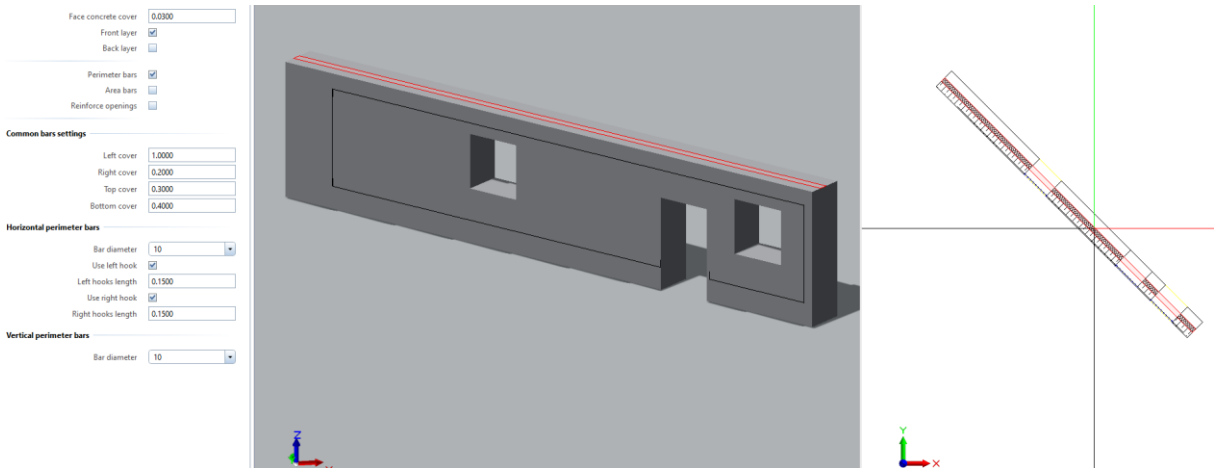

Рисунок 4.2.5 – Периметральне армування при нетипічних відступах з боків

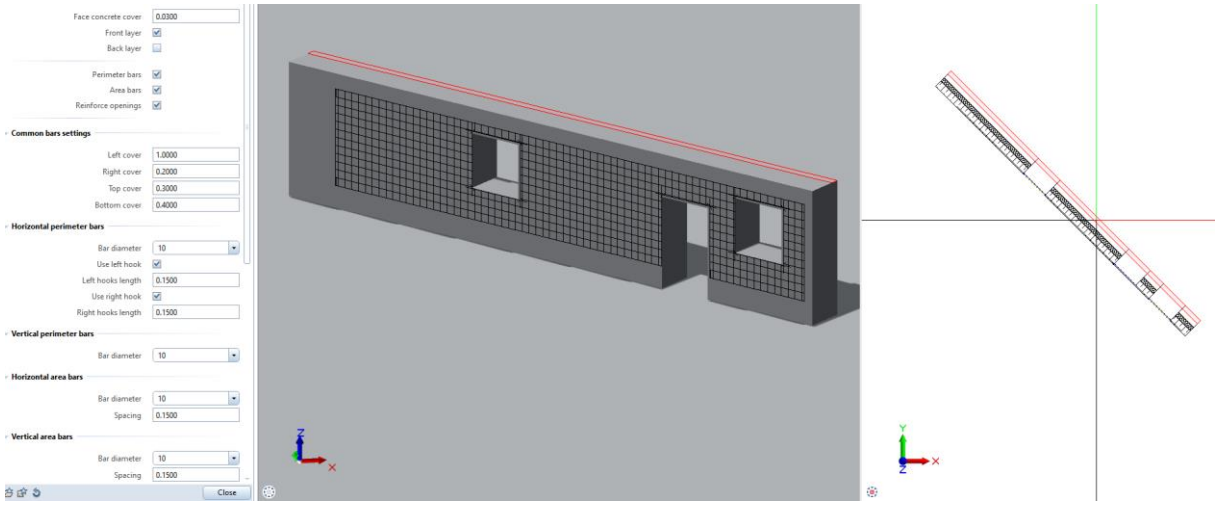

Рисунок 4.2.6 – Армування всіх типів при нетипічних відступах з боків

Якщо брати окремі типи армувань, то вони досить зрозумілі на інтуїтивному рівні.

Щоб створити периметральне армування потрібно вказати діаметри горизонтальних та вертикальних стержнів, а також налаштувати довжину гачків у горизонтальної арматури (рис. 4.2.7).

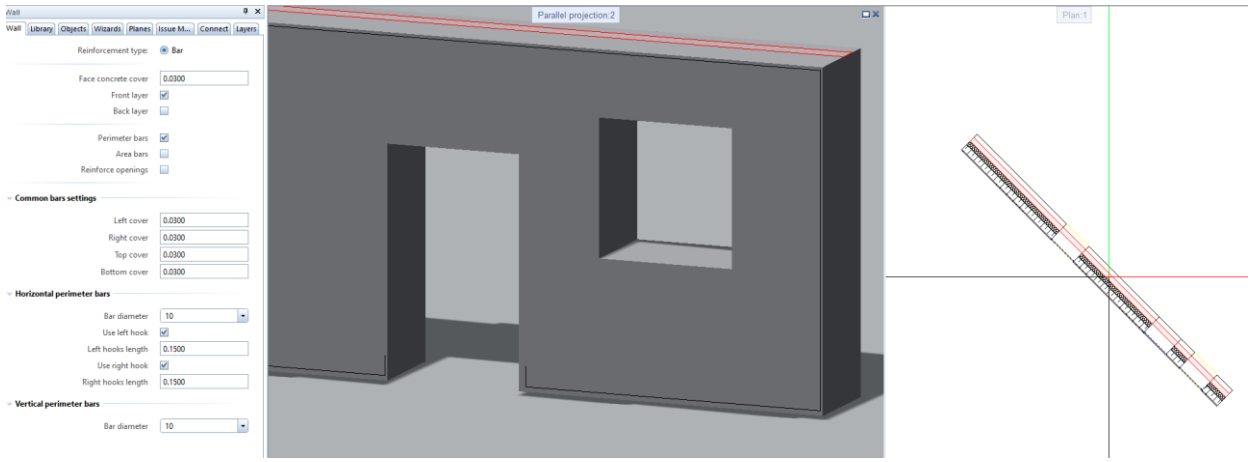

Рисунок 4.2.7 – Периметральне армування

Для створення армування по площі знадобиться вказати тільки діаметр для горизонтальних і вертикальних стержнів та значення проміжку між стержнями для кожного напрямку. Для наочності на рисунках 4.2.8 та 4.2.9 створені армування з різними відношеннями відстаней для стержнів.

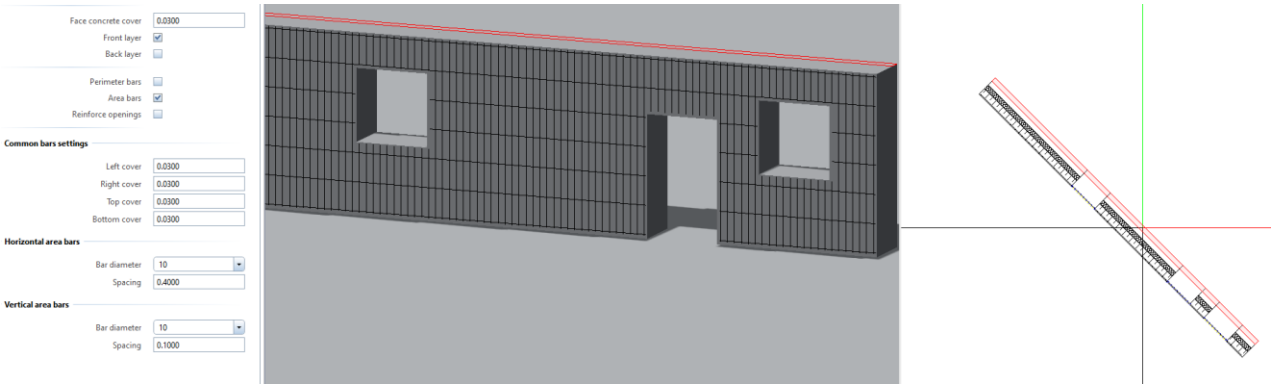

Рисунок 4.2.7 – Армування по площі, де відстань між горизонтальними стержнями більша, ніж відстань між вертикальними

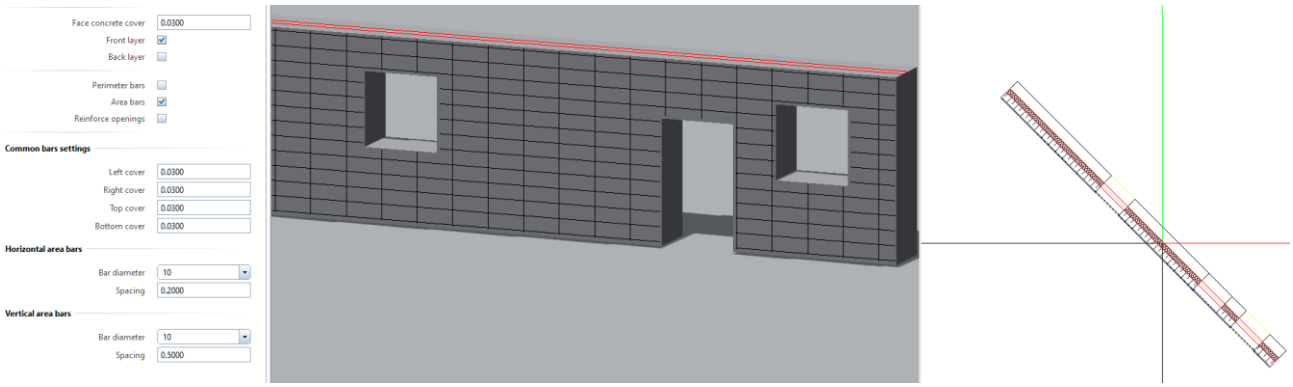

Рисунок 4.2.8 – Армування по площі, де відстань між горизонтальними стержнями менша, ніж відстань між вертикальними

Для армування всіх вирізів користувач має задати відстані від краю, діаметри стержнів, довжини на потребність в гачках, а також окремо довжину та відступ від краю для діагональних стержнів. На рисунку 4.2.9 зображено армування тільки вирізів з гіперболізованими довжинами гачків.

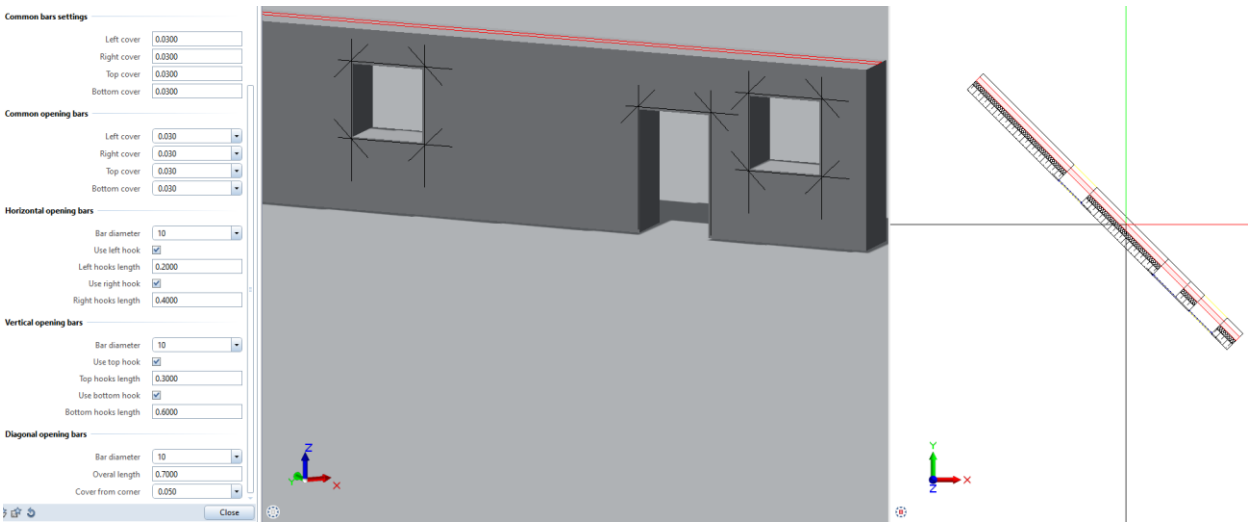

Рисунок 4.2.9 – Армування вирізів з нестандартними параметрами

4.3. Висновки до розділу

Розглянуто процес роботи користувача з Allplan системою, зокрема PythonParts модулем. Показана робота з палеткою параметрів.

ВИСНОВКИ

У даній роботі було розглянуто проблематику автоматизації PythonParts модулів для BIM системи Allplan.

1. Проведено огляд конкуруючих застосунків, який показав актуальність розробки модуля PythonParts.

2. Проведено аналіз засобів програмної розробки, на основі якого обгрунтовано використання Visual Studio 2017, вбудованого в неї дебагера PTVSD та мови програмування Python з .pyd модулями.

3. Розроблено алгоритми армування та пошуку точок за допомогою яких побудовано модель для створення армування стін.

4. Проведено реалізацію запропонованих алгоритмів у вигляді PythonPart модуля, що надало користувачу можливість армування у вигляді периметрального, по площі та армування вирізів у відповідності до параметрів користувача.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Что такое данные Revit? [Електронний ресурс] – Режим доступу до ресурсу: https://pro.arcgis.com/ru/pro-app/2.6/help/data/revit/what-is-revit-data-.htm.

2. Structural BIM Software [Електронний ресурс] – Режим доступу до ресурсу: https://www.tekla.com/products/tekla-structures.

3. Програмування числових методів мовою Python : підруч. / А. В. Анісімов, А. Ю. Дорошенко, С. Д. Погорілий, Я. Ю. Дорогий; за ред. А. В. Анісімова. – К. : Видавничо-поліграфічний центр "Київський університет", 2014. – 640 с

4. Отличия между .pyc, .pyd и .pyo файлами [Електронний ресурс]. – 2018. – Режим доступу до ресурсу: https://www.severcart.ru/blog/all/pyc_pyd_pyo/.

5. Microsoft Visual Studio Documentation [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/visualstudio/get-started/visual-studio-ide?view=vs-2019.

6. IntelliSense [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://code.visualstudio.com/docs/editor/intellisense#_intellisense-for-your-programming-language.

7. PIP ptvsd module [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://pypi.org/project/ptvsd/#:~:text=ptvsd%204.3.,2&text=The%20Python%20Visual%20Studio%20Debugger,tracker%20is%20hosted%20on%20GitHub.

8. Extensible Markup Language (XML) [Електронний ресурс]. – 2006. – Режим доступу до ресурсу: https://www.w3.org/TR/2006/REC-xml11-20060816/.

9. Documentation Allplan Python API [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://pythonparts.allplan.com/.

ДОДАТОК А

Створення параметричних металоконструкцій в CAD системі PythonParts

Специфікація

УКР.НТУУ«КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ТМ72

Аркушів 2

Київ – 2021

Позначення	Найменування	Примітки
Документація		
УКР.НТУУ«КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ТМ72	Коваль Я.В._ТМ72.docx	Пояснювальна записка
Компоненти		
УКР.НТУУ«КПІ ім. Ігоря Сікорського».ТМ72	WallReinforcementInteractor.py	Програмна реалізація модуля
УКР.НТУУ«КПІ ім. Ігоря Сікорського».ТМ72	WallReinforcementInteractor.pal	Розмітка палетки для параметрів
УКР.НТУУ«КПІ ім. Ігоря Сікорського».ТМ72	WallReinforcementInteractor.png	Іконка модуля
УКР.НТУУ«КПІ ім. Ігоря Сікорського».ТМ72	WallReinforcementInteractor.pyp	Ініціалізатор для палетки

ДОДАТОК Б

Створення параметричних металоконструкцій в CAD системі PythonParts

Текст програми

УКР.НТУУ«КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ТМ72

Аркушів 10

Київ – 2021

```
def create_interactor(coord_input, pyp_path, str_table_service):
    """
    Create the interactor

    Args:
        coord_input:        coordinate input
        pyp_path:           path of the pyp file
        str_table_service:  string table service
    """

    return WallReinforcementInteractor(coord_input, pyp_path, str_table_service)



class WallReinforcementInteractor():

    def __init__(self, coord_input, pyp_path, str_table_service):
        self.coord_input            = coord_input
        self.doc                    = coord_input.GetInputViewDocument()
        self.pyp_path               = pyp_path
        self.str_table_service      = str_table_service
        self.first_point_input      = True
        self.is_closed              = False
        self.tiers_count            = None
        self.z_angle                = AllplanGeo.Angle()
        self.wall_orc            = None
        self.window_orchestrators   = []
        self.door_orchestrators     = []
        self.bending_roller         = 1
        self.steel_grade            = 1
        self.concr_grade            = 1

        self.first_point            = AllplanGeo.Point3D()
        self.coord_input.InitFirstElementInput(AllplanIFW.InputStringConvert("Select the wall"))

        type_query = AllplanIFW.QueryTypeID(AllplanElementAdapter.WallTier_TypeUUID)
        sel_query = AllplanIFW.SelectionQuery(type_query)
        self.element_filter = AllplanIFW.ElementSelectFilterSetting(sel_query, True)
```

```
        self.show_palette()

    def process_mouse_msg(self, mouse_msg, pnt, msg_info):
        self.coord_input.SelectElement(mouse_msg, pnt, msg_info, False, True, True, self.element_filter)

        wall_tier_plane = self.coord_input.GetSelectedElement()

        if wall_tier_plane.IsNull():
            return True

        error, polygon, normal_vec = self.coord_input.SelectWallFace(wall_tier_plane, pnt, True)

        if self.is_closed:
            return True

        if self.coord_input.IsMouseMove(mouse_msg):
            self.create_preview(wall_tier_plane)
        else:
            self.create_elements(wall_tier_plane)
            self.is_closed = True

        return True

    def create_reinforcement(self, wall_tier_plane):
        wall = AllplanElementAdapter.BaseElementAdapterParentElementService.GetParentElement(wall_tier_plane)

        wall_childs = AllplanElementAdapter.BaseElementAdapterChildElementsService.GetChildModelElementsFromTree(wall)

        wall_tiers = []
        for wall_child in wall_childs:
            if wall_child == AllplanElementAdapter.WallTier_TypeUUID:
                wall_tiers.append(wall_child)

        if len(wall_tiers) < 1:
            return True
```

```
        self.tiers_count = len(wall_tiers)

        self.set_wall_orchestrator(wall_tiers)
        self.set_openings_orchestrators(wall_tier_plane)

        placements = []

        if self.build_ele.ReinforcementType.value == 0:
            if self.build_ele.UsePerimeterBars.value == True:
                placements += self.create_perimeter_reinforcement()
            if self.build_ele.UseAreaBars.value == True:
                placements += self.create_area_reinforcement()
            if self.build_ele.IsOpeningsReinforced.value == True:
                placements += self.create_openings_reinforcement()

        elif self.build_ele.ReinforcementType.value == 1:
            self.create_mesh_reinforcement()

        return placements

    def set_wall_orchestrator(self, wall_tiers):
        first_model_geometry = wall_tiers[0].GetModelGeometry()
        first_layer_vertices = []
        last_model_geometry = wall_tiers[-1].GetModelGeometry()
        last_layer_vertices = []

        self.wall_orc = ModelsOrchestrator(first_model_geometry, last_model_geometry)
        self.apply_face_concrete_cover()
        self.apply_sides_concrete_covers()

        return True

    def create_preview(self, wall_tier_plane):
        reinforcement = self.create_reinforcement(wall_tier_plane)

        AllplanBaseElements.DrawElementPreview(self.doc, AllplanGeo.Matrix3D(), reinforcement, False,
None)
```

```
        return True


class ModelsOrchestrator():
    def __init__(self, polyhedron3D_1, polyhedron3D_2):
        #orchestrator divided to face and back points
        self.face_wall_bottom_points    = None
        self.back_wall_bottom_points    = None
        self.face_wall_top_points       = None
        self.back_wall_top_points       = None
        #Normales
        self.top_normal     = None
        self.bottom_normal  = None
        #       *
        #  ---->*       left normal
        #       *
        self.left_normal    = None
        #       *
        #       *<----- right normal
        #       *
        self.right_normal   = None
        self.face_normal    = None
        self.back_normal    = None

        vertices1_bottom, vertices1_top = self.get_extreme_points(polyhedron3D_1)
        if vertices1_bottom == []:
            return
        vertices2_bottom, vertices2_top = self.get_extreme_points(polyhedron3D_2)
        if vertices2_bottom == []:
            return

        poly1_lines_bottom  = self.create_sorted_lines_from_vertices(vertices1_bottom)
        poly1_lines_top     = self.create_sorted_lines_from_vertices(vertices1_top)
        poly2_lines_bottom  = self.create_sorted_lines_from_vertices(vertices2_bottom)
        poly2_lines_top     = self.create_sorted_lines_from_vertices(vertices2_top)

        #----------- take this
```

```
    #-----------
    #
    #-----------
    #----------- and this
    bottom_vertices     = self.find_couple_of_longest_dist(poly1_lines_bottom, poly2_lines_bottom)
    top_vertices        = self.find_couple_of_longest_dist(poly1_lines_top, poly2_lines_top)

    self.set_points_to_walls(bottom_vertices, top_vertices)

    self.set_normales()

def get_extreme_points(self, poly3D):
    vertices = sorted(list(poly3D.GetVertices()), key = lambda point: (point.Z))
    if len(vertices) < 8:
        return [], []
    high_vertices = vertices[-4:]
    low_vertices = []
    for v in vertices:
        for hv in high_vertices:
            if v.X == hv.X and v.Y == hv.Y:
                low_vertices.append(v)
        if len(low_vertices) == 4:
            break

    return low_vertices, high_vertices

def create_sorted_lines_from_vertices(self, vertices):
    lines = []
    dist = vertices[0].GetDistance(vertices[1])
    min_dist = dist
    max_dist = dist

    for i in range(2, len(vertices)):
        dist = vertices[0].GetDistance(vertices[i])
        if dist > max_dist:
            max_dist = dist
        if dist < min_dist:
```

```
            min_dist = dist

    for i in range(1, len(vertices)):
        dist = vertices[0].GetDistance(vertices[i])
        if dist > min_dist and dist < max_dist:
            lines.append([vertices[0], vertices[i]])
            line2 = []
            for j in range(1, len(vertices)):
                if j != i:
                    line2.append(vertices[j])
            lines.append(line2)
            break

    sorted_lines = []
    sorted_lines.append(sorted(lines[0], key = lambda point: (point.X, -point.Y)))
    sorted_lines.append(sorted(lines[1], key = lambda point: (point.X, -point.Y)))
    return sorted_lines

def find_couple_of_longest_dist(self, lines1, lines2):
    v1_index = 0
    v2_index = 0

    max_dist = 0
    for i in range(len(lines1)):
        for j in range(len(lines2)):
            dist = lines1[i][0].GetDistance(lines2[j][1])
            if dist > max_dist:
                max_dist = dist
                v1_index = i
                v2_index = j

    return lines1[v1_index], lines2[v2_index]

def set_points_to_walls(self, bottom_tuple, top_tuple):
    if bottom_tuple[0][0].Y < bottom_tuple[1][0].Y:
        self.face_wall_bottom_points = bottom_tuple[0]
        self.back_wall_bottom_points = bottom_tuple[1]
```

```
        else:
            self.back_wall_bottom_points = bottom_tuple[0]
            self.face_wall_bottom_points = bottom_tuple[1]

        if top_tuple[0][0].Y < top_tuple[1][0].Y:
            self.face_wall_top_points = top_tuple[0]
            self.back_wall_top_points = top_tuple[1]
        else:
            self.back_wall_top_points = top_tuple[0]
            self.face_wall_top_points = top_tuple[1]


    def set_normales(self):
        self.top_normal     = AllplanGeo.Vector3D(0, 0, -1)
        self.bottom_normal  = AllplanGeo.Vector3D(0, 0, 1)
        self.left_normal                    =     AllplanGeo.Vector3D(self.face_wall_bottom_points[0],
self.face_wall_bottom_points[1])
        self.face_normal                    =     AllplanGeo.Vector3D(self.face_wall_bottom_points[0],
self.back_wall_bottom_points[0])
        self.left_normal.Normalize()
        self.face_normal.Normalize()
        self.right_normal   = self.left_normal * -1
        self.back_normal    = self.face_normal * -1

    def apply_face_cover(self, face_cover):
        for point in self.face_wall_bottom_points:
            point.Set(point + self.face_normal * face_cover)
        for point in self.face_wall_top_points:
            point.Set(point + self.face_normal * face_cover)
        for point in self.back_wall_bottom_points:
            point.Set(point + self.back_normal * face_cover)
        for point in self.back_wall_top_points:
            point.Set(point + self.back_normal * face_cover)

    def apply_covers(self, left_cover, right_cover, top_cover, bottom_cover, is_inversed = False):
        self.apply_left_cover(left_cover, is_inversed)
        self.apply_right_cover(right_cover, is_inversed)
        self.apply_top_cover(top_cover, is_inversed)
```

```
    self.apply_bottom_cover(bottom_cover, is_inversed)

def apply_left_cover(self, left_cover, is_inversed = False):
    if is_inversed == False:
        normal = self.left_normal
    else:
        normal = self.right_normal

    self.face_wall_bottom_points[0].Set(self.face_wall_bottom_points[0] + normal * left_cover)
    self.face_wall_top_points[0].Set(self.face_wall_top_points[0] + normal * left_cover)
    self.back_wall_bottom_points[0].Set(self.back_wall_bottom_points[0] + normal * left_cover)
    self.back_wall_top_points[0].Set(self.back_wall_top_points[0] + normal * left_cover)

def apply_right_cover(self, right_cover, is_inversed = False):
    if is_inversed == False:
        normal = self.right_normal
    else:
        normal = self.left_normal

    self.face_wall_bottom_points[1].Set(self.face_wall_bottom_points[1] + normal * right_cover)
    self.face_wall_top_points[1].Set(self.face_wall_top_points[1] + normal * right_cover)
    self.back_wall_bottom_points[1].Set(self.back_wall_bottom_points[1] + normal * right_cover)
    self.back_wall_top_points[1].Set(self.back_wall_top_points[1] + normal * right_cover)

def apply_top_cover(self, top_cover, is_inversed = False):
    if is_inversed == False:
        normal = self.top_normal
    else:
        normal = self.bottom_normal

    for point in self.face_wall_top_points:
        point.Set(point + normal * top_cover)
    for point in self.back_wall_top_points:
        point.Set(point + normal * top_cover)

def apply_bottom_cover(self, bottom_cover, is_inversed = False):
    if is_inversed == False:
```

```
            normal = self.bottom_normal
        else:
            normal = self.top_normal

        for point in self.face_wall_bottom_points:
            point.Set(point + normal * bottom_cover)
        for point in self.back_wall_bottom_points:
            point.Set(point + normal * bottom_cover)

    def get_angle(self):
        return                AllplanGeo.Vector2D(AllplanGeo.Point2D(self.face_wall_bottom_points[0].X,
self.face_wall_bottom_points[0].Y),
                                    AllplanGeo.Point2D(self.face_wall_bottom_points[1].X,
self.face_wall_bottom_points[1].Y)).GetAngle()
```

ДОДАТОК В

Створення параметричних металоконструкцій в CAD системі PythonParts

Опис програмного продукту

УКР.НТУУ«КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ТМ72

Аркушів 9

Київ – 2021

АНОТАЦІЯ

Даний додаток містить опис модуля для створення для створення армування стіни в CAD системі PythonParts. Створений програмний продукт може отримувати параметри з палетки користувача та виконувати наступні функції:

- Створювати 2 шари (передній та задній);
- Окремо створювати периметральне армування, армування по площі та армування вирізів;
- Створювати модель для попереднього перегляду армування;

Модуль було написано на мові програмування Python, використовуючи прикладну технологію PythonParts, що дозволяє поєднувати алгоритми обчислення армування та палетку, з якої беруться параметри.

ЗМІСТ

1. ЗАГАЛЬНІ ВІДОМОСТІ

У цьому додатку міститься опис PythonPart модуля для армування стін у середовищі Allplan. Додаток Б складається з коду реалізації алгоритму.

Для роботи з модулем потрібне передвстановлене середовище Allplan, у проектні файли якого потрібно додати код модуля.

При розробці використовувалася IDE Visual Studio 2017, редактор коду VS Code, мова програмування Python та влаштована в додаток Allplan система PythonParts.

2. ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Розроблене програмне забезпечення дозволяє створити модель армування стіни при наведенні на неї курсора за допомогою PythonPart середовища, яке включає в себе як звичайні математичні функції, так і специфічні методи для роботи з застосунком Allplan.

Модуль має наступний функціонал:

- може мати 2 шари (передній та задній) ;
- бетонний відступ від лицевої сторони кожного шару;
- можливість окремо створювати периметральне армування, армування по площі та окреме армування вирізів;
- можливість додати відступи від сторін – зверху, знизу, зліва та справа для усієї конструкції;
- можливість задання параметрів для периметрального армування (діаметр стержня, довжину гачків для горизонтального армування) ;
- можливість задання параметрів для армування по площі (діаметр стержня, проміжок між стержнями) ;
- можливість задання параметрів відступу для армування вирізів (зверху, знизу, зліва та справа) ;
- можливість задання параметрів армування вирізів (діаметр стержня, довжину лівих, правих, верхніх та нижніх гачків) ;
- можливість задання параметрів діагонального армування вирізів (діаметр стержня, довжину стержня та відступ від краю) ;
- можливість попереднього перегляду моделі до створення армування
- можливість створення моделі армування.

Функціонал модуля обмежується тільки специфічними по формі стінами.

3. ОПИС ЛОГІЧНОЇ СТРУКТУРИ

Так як модуль повинен бути створений в середовищі Allplan в підсистемі PythonParts, то обов’язковим є використання мови Python для реалізації алгоритмів. Також для спрощення розробки потрібно використовувати Visual Studio версії 2017 року або нижче.

Для реалізації алгоритмів обробки початкових даних використовуються .pyd модулі, що дають змогу взаємодіяти всім елементам системи, від створення та виводу палетки, до обробки нормалей та площин.

Розроблений модуль працює у застосунку Allplan будь-якої версії. Для роботи з ним потрібно створити стіну та задати параметри палетки.

4. ВИКОРИСТОВУВАНІ ТЕХНІЧНІ ЗАСОБИ

Для роботи з модулем потрібне передналаштоване середовище Allplan будь-якої версії.

Мінімальні системні вимоги для застосунка 2021 року:

– Intel процесор з підтримкою AVX
– гб оперативної пам’яті
– 10 гб на жорсткому диску
– OpenGL 4.2 сумісна відеокарта, з 2 гб пам’яті та розширенням 1280x1024
– Операційна система Windows 8.1 або Windows Server 2012 R2

5. ВИКЛИК І ЗАВАНТАЖЕНЯ

Якщо модуль використовується у режимі розробки, то потрібно кореневі файли додати до папок застосунка Allplan. Файл з розширенням .py додати у папку по шляху .\Nemetschek\Allplan\2021\ETC\PythonPartsScripts\WallReinfPythonPart, а файли з розширеннями .pal, .png та .pyp додати у папку по шляху Data\Allplan\Allplan 2021\Std\Library\WallReinforcement.

Для виклику модуля після додання відповідних файлів по місцям, як і при доданні модуля у релізну версію Allplan, користувач має змогу знайти в бібліотеці модулів поточний.

Для відкриття палетки потрібно перетягнути іконку модуля на програмне вікно, де знаходиться стіна. При натисненні на стіну сформується готове армування, що буде відповідати параметрам палетки.

6. ВХІДНІ І РЕЗУЛЬТУЮЧІ ДАНІ

Вхідними даними є параметри стіни, на яку наведений курсор, а саме всі вершини.

Результуючими даними є об’єкти, які зберігають відсортовані верхні та нижні крайні лінії для кожного елементу та кожного шару. Сюди ж, до вихідних даних, відносяться стержні для армування та інтерактор, який містить у собі дані про поточне положення курсора разом з параметрами палетки.

ДОДАТОК Г

Створення параметричних металоконструкцій в CAD системі PythonParts

Акт впровадження

УКР.НТУУ«КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ТМ72

Аркушів 9

Київ – 2021

Затверджую

директор ТОВ

«АЛЛБАУ СОФТВЕР»

_____ Смирнов. Ю.О.

Акт

впровадження результатів дипломної роботи

Коваля Ярослава Валерійовича

«Створення параметричних металоконструкцій в CAD системі PythonParts»

Вх. н. 3 Дата 25.05.2021

Директор ТОВ «АЛЛБАУ СОФТВЕР» склав поточний документ про те, що результати дипломної роботи студента Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського», представленої на захист дипломного проекту, розглянуті для використання у ТОВ «АЛЛБАУ СОФТВЕР» в якості програмного забезпечення для автоматизації армування стін.

Результати дипломної роботи мають практичне значення, є актуальними та потребують подальшого розвитку.

Директор 25.05.2021 _____ Смирнов. Ю.О.

(дата, підпис)